University of Alberta

ANALOG ITERATIVE ERROR CONTROL DECODERS

by

Chris Winstead ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Department of Electrical and Computer Engineering

Edmonton, Alberta
Spring 2005

0-494-08314-X

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN
Our file Notre reference
ISBN

NOTICE:
The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:
L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Abstract

All communication and storage media are subject to corruption of their digital information. Error control codes, in principle, permit communication and storage of digital information with arbitrarily low probability of error. Iterative error control decoding algorithms provide error protection close to theoretical limits. Conventional implementations of these algorithms, using digital circuits, are extremely complex. Analog computation has been studied as an alternative approach for implementing iterative decoders.

Iterative decoders use a large-scale, parallel network of "soft" probability calculators. These nodes exchange probability information with each other. After many iterations, local information (individual noisy samples) is transformed into global information (an estimate of the word as a whole). These local fuzzy operations map naturally to a family of simple analog circuits.

In this thesis, we examine new circuit topologies and design principles for implementing analog decoders using CMOS technology. We introduce novel circuit structures for reducing complexity, and for minimum power supply voltage. We examine issues in performance and scaling which are unique to analog decoding circuits. We report results for the first successful analog decoder to be implemented in CMOS technology. We also present the design of and test results for an analog Turbo product decoder with a coded block length of 256, which is the largest and most powerful analog iterative decoder to date.

Preface

This thesis presents the results of work which began at the University of Utah in 2000, and continued at the University of Alberta between 2002 and 2004. The author has contributed to the design of seven analog decoder circuits fabricated on five chips. These chips included the first successful CMOS implementation of an analog MAP decoder. The author has also designed an analog Turbo Product decoder, which is the largest and most powerful analog decoder design to have been attempted as of the writing of this thesis.

The author's original work includes new results on the performance of analog decoders under the influence of non-ideal analog effects, and resultant design principles for optimizing the performance of analog decoders. Also among the author's contributions are novel circuit topologies for implementing analog decoders, including an approach which allows low supply voltage.

This thesis presents a nearly complete presentation of concepts and vocabulary which are required for a clear statement of the author's results. In many instances, the terminology or definitions in the existing literature are not quite adequate. The work of this thesis requires a synthesis of code construction, decoding techniques, and circuit design. The designs are built upon concepts which are not necessarily well-known to the audience of this thesis, and the source literature for these concepts occasionally contain minor errors which cause confusion to new readers.

To avoid a clumsy collision of vocabulary, and to rescue some readers from obscure or difficult source literature, the author has presented the necessary concepts under a unified vocabulary, which is outlined in the introductory chapters. Several examples are given throughout the early chapters. The reader should be aware that, in most instances, the examples represent portions of the author's designs. This apparently verbose approach in early chapters allows for a more direct and unambiguous exhibition of results in later chapters.

Throughout this thesis, whenever "speed" or "throughput" measurements are reported, they reference the rate of *information* bits per second. When throughput is cited from other works, the author has converted the data where necessary to units of information bits per second. Energy efficiency is also reported in units of Joules per information bit.

In order to clearly indicate the author's work, sections which present mostly original contributions are marked with an asterisk ('*'). In some cases an entire chapter contains mostly original work, and is also indicated by an asterisk.

Acknowledgements

Many individuals made significant contributions to the ideas developed in this thesis. Our study of analog decoding began at the University of Utah, in partnership with Christian Schlegel, Chris Myers, Reid Harrison, Jie Dai, Scott Little and Shuhuan Yu. The success of the first CMOS analog decoder (presented in Chapter 8 of this thesis) is due to their influence and participation. Many design decisions for the Product decoder (presented in Chapter 10 of this thesis) were also influenced by Jie Dai and Shuhuan Yu.

At the University of Alberta, this work was assisted by the additional supervision of Vincent Gaudet, who provided invaluable assistance during the layout, fabrication and testing phases. Nhan (Dave) Nguyen collaborated on the design of the test interface, which was used to obtain most of the results reported in this thesis. Thanks are also owed to Sheryl Howard and David Haley for their comments on designs and test procedures, and for reviewing portions of this thesis.

I also extend my thanks to Anthony Rapley, Mimi Yiu and Siavash Sheikh Zeinoddin for their friendship, and for many stimulating discussions of communication systems and iterative decoders.

Funding for the University of Utah project was provided by NSF grant CCR-9971168. At the University of Alberta, the project was continued under funding from the Alberta Informatics Circle of Research Excellence (iCORE). Design software and fabrication services were provided by the Canadian Microelectronics Corporation.

Contents

1 Introduction 1
- 1.1 History of analog decoding. 4
  - 1.1.1 Physical demonstrations. 8
  - 1.1.2 Development of design principles. 12
- 1.2 Contributions of this thesis. 18
  - 1.2.1 Contributions to coding theory. 19
  - 1.2.2 A novel "reference input" sum-product topology. 19
  - 1.2.3 A novel low-voltage sum-product topology. 20
  - 1.2.4 Mismatch in analog decoders. 20
  - 1.2.5 Comparator offsets and yield. 21
  - 1.2.6 The first CMOS analog decoders. 22
  - 1.2.7 A length-16 analog MAP decoder. 23
  - 1.2.8 A length-256 Block Turbo decoder. 24
- 1.3 Outline of this thesis. 24

2 Error-Control Codes and Decoders 27
- 2.1 Communications systems. 27
  - 2.1.1 Probabilities and log-likelihood ratios. 28
- 2.2 The Shannon limit. 30
- 2.3 Linear binary block codes. 32
  - 2.3.1 Factor graphs and Normal graphs. 34
  - 2.3.2 Tanner graphs. 36
- 2.4 Decoding algorithms. 38
  - 2.4.1 MAP versus ML decoding. 38
  - 2.4.2 The sum-product algorithm. 39
- 2.5 Good codes. 41
  - 2.5.1 Parallel Concatenated Convolutional Codes: Turbo Codes. 42
  - 2.5.2 Serially Concatenated Convolutional Codes. 45
  - 2.5.3 Repeat-Accumulate codes. 47
  - 2.5.4 Low-Density Parity Check codes. 48
  - 2.5.5 Block Turbo Codes. 49
  - 2.5.6 Summary 52

3 Trellis Codes 55
- 3.1 Trellis graphs for block and convolutional codes. 55
- 3.2 Soft-information trellis decoding. 59
  - 3.2.1 MAP decoding on trellises. 59

3.2.2 Transition matrices 60
3.2.3 Details of the BCJR algorithm. 61
3.2.4 An example decoder: (8,4) tailbiting Hamming trellis. . . . 62
3.3 Construction of block code trellises. 67
3.3.1 Generator matrices and the trellis product. 67
3.3.2 Construction of tailbiting trellises. 69
3.3.3 Squaring construction for Hamming and Reed-Muller codes. 71
3.3.4 Example: (16,11) Hamming code construction. 73
3.3.5 *Tailbiting squaring construction. 78
3.4 *True-MAP decoding on tailbiting trellises. 81
3.4.1 *Complexity of the exact MAP algorithm. 83
3.4.2 *Exact MAP for tailbiting (8,4) Hamming trellis. 83

4 LDPC Codes 87
4.1 Regular and Irregular Ensembles 87
4.1.1 Regular LDPC codes 87
4.1.2 Irregular LDPC Codes 89
4.2 Density Evolution and Code Selection 91
4.3 The AWGN channel and the Gaussian Approximation. 93
4.3.1 Computing the threshold. 95
4.3.2 Threshold Determination 97
4.3.3 *Numerical methods for density evolution. 101
4.3.4 *Iterated integration. 105
4.4 An algorithm for computing the threshold. 106

5 Analog Decoding Circuits 111
5.1 The translinear principle. 111
5.1.1 Basic translinear circuits. 113
5.1.2 The Gilbert vector multiplier. 115
5.1.3 Translinear sum-product circuits. 116
5.1.4 Duality in translinear circuits. 117
5.2 CMOS translinear circuits. 118
5.2.1 MOS device models. 120
5.2.2 The canonical CMOS sum-product circuit. 123
5.2.3 Translinear analysis 125
5.2.4 Renormalization of current vectors. 126
5.3 * Supply voltage in canonical circuits 127
5.3.1 Minimum allowable supply voltage. 129
5.3.2 Approximations. 130
5.4 *A reduced-complexity sum-product circuit. 131
5.4.1 *Complexity of the reference circuit. 132
5.4.2 *Implementing all directions. 134

6 *Low-voltage Analog Decoding Circuits 139
6.1 Eliminating V_{ref} 139
6.2 Low-voltage sum-product circuits 140

6.2.1 A general low-voltage sum-product topology 142
6.2.2 Supply voltage in low-voltage circuits 142
6.2.3 Approximations. 145
6.2.4 Process scaling and temperature effects. 145
6.2.5 Renormalization 147
6.3 Decoder architectures 151
6.3.1 Trellis decoders 151
6.3.2 LDPC (Tanner Graph) Decoders 153
6.4 An example decoder 154

7 *Scaling and Performance in Analog Iterative Decoders. 159
7.1 Interface architecture. 160
7.1.1 Sample-and-hold input interfaces. 161
7.1.2 Comparators and yield. 167
7.2 Mismatch. 170
7.2.1 Modeling mismatch in analog sum-product circuits. 171
7.2.2 Feed-forward analysis. 174
7.2.3 Density evolution analysis of lateral effects. 175
7.2.4 Comparison of feed-forward and lateral mismatch effects. . 178

8 *A CMOS Analog Decoder for an (8,4) Tailbiting Hamming Code. 181
8.1 The analog sum-product components. 181
8.1.1 The Tree circuit. 182
8.1.2 Trellis section one. 182
8.1.3 Trellis section two. 183
8.1.4 T_1 (out). 183
8.1.5 T_2 (out). 189
8.2 Interfaces. 190
8.2.1 S/H input circuits. 192
8.2.2 Comparator circuit. 192
8.3 Physical test results. 193
8.3.1 Dynamics of the decoder. 196
8.3.2 Measurements in strong inversion. 197
8.3.3 Measurements in weak inversion. 198
8.3.4 Mixed-signal interference. 199

9 *A CMOS Analog Decoder for a (16,11) Hamming Code. 203
9.1 The analog sum-product components. 204
9.1.1 Subtrellis implementations. 209
9.1.2 Reset switches. 209
9.2 Interfaces. 210
9.2.1 S/H input circuits. 214
9.2.2 Comparator circuit. 214
9.3 Test interface. 216
9.3.1 Hardware. 216
9.3.2 Software. 217

9.3.3 Noise in the test interface. 218
9.3.4 Loop-back interface test. 219
9.3.5 Discussion of the interfaces. 220
9.4 Characteristics of the decoder. 221
9.4.1 Dynamics. 222

10 *An Analog $(16,11)^2$ Turbo Product Decoder. 227
10.1 Design of the decoder. 228
10.1.1 Floorplan and interleaving. 228
10.1.2 Scalability. 228
10.2 Characteristics of the decoder. 229
10.2.1 Speed. 232
10.2.2 Performance. 232

11 Conclusions and Outlook. 237

List of Tables

1.1 A summary of published digital iterative decoder designs. 13

1.2 A summary of published analog iterative decoder designs. 14

2.1 List of notation for communications and coding. 30

3.1 RM (ρ, ν) codes and their partition degrees. 75

4.1 Exact (SNR^*) [69] and Gaussian-approximation (SNR^*_{GA}) thresholds for various rate-$1/2$ degree distributions on the AWGN channel, in terms of E_b/N_0 (dB). The BPSK limit for this rate is 0.1870dB. 99

4.2 Exact (SNR^*) [69] and Gaussian-approximation (SNR^*_{GA}) thresholds for various rate-$1/2$ degree distributions on the AWGN channel, in terms of E_b/N_0 (dB). The BPSK limit for this rate is 0.1870dB. 100

4.3 Estimated parameters, bias (μ_s), error standard deviation (σ_s), and threshold results for various regular LDPC ensembles. In all cases, the tolerance is $\sigma_y = 0.001$ and $\Delta r = 0.025$. 104

8.1 Summary of (8,4) Hamming decoder characteristics. 194

9.1 Summary of (16,11) Hamming decoder characteristics. Speed and power figures are estimated from simulations, and are not within the range of accurate measurement for the test interface. "Tested Speed" refers to the maximum speed of the test interface. 226

10.1 Summary of $(16,11)^2$ Product decoder characteristics. Speed and power measurements are estimated based on simulations, but were outside the accurately measurable range of available test equipment. "Tested speed" refers to the maximum reliable test speed of our interface. 236

List of Figures

1.0.1 Decoder size vs performance, comparing a collection of synthesized digital decoders in a 0.5μm CMOS process [94]. Two analog decoders are shown as labeled. The Product decoder is shown adjusted for the difference in process feature size. The analog decoders are reported in this thesis. 5

1.0.2 Decoder power vs performance, showing a collection of simulated digital decoders [94] alongside two fabricated analog decoders. The analog decoders are labeled as an (8,4) Hamming decoder and a $(16,11)^2$ Product decoder. Results for the analog decoders are reported in this thesis. 6

1.1.1 Energy expense per bit for analog vs digital decoders. The circles represent reported digital decoders, and the stars represent fabricated analog decoders. 12

2.1.1 Communications system model. 28

2.2.1 Shannon and BPSK limits. 32

2.3.1 Example factor graph for $f(x,y,z) = f_1(x,y) \cdot f_2(x,z) \cdot f_3(y) \cdot f_4(y,z)$. 34

2.3.2 Factor graph with hidden variables. 35

2.3.3 Normal graph corresponding to Figure 2.3.1. 35

2.3.4 Normal graph corresponding to Figure 2.3.2. 35

2.3.5 Example Tanner graph. 37

2.3.6 Normalized Tanner graph corresponding to Figure 2.3.5. 37

2.4.1 Hidden variable insertion. 39

2.4.2 Function node for a Boolean constraint on three variables. 40

2.4.3 Implementation of probability propagation in a node of degree three. 41

2.5.1 PCCC Turbo Code encoder. 43

2.5.2 Decoder for a PCCC Turbo Code. 43

2.5.3 Factor graph for a PCCC Turbo Code. 44

2.5.4 Performance of the original rate-1/2 PCCC Turbo Code, with length 65536 and 18 decoding iterations [15]. The BPSK limit is also indicated. 45

2.5.5 Encoder for an SCCC Turbo Code. 45

2.5.6 Decoder structure for SCCC Turbo Codes. 46
2.5.7 Factor graphs for SCCC Turbo Codes. 46
2.5.8 Repeat-Accumulate encoder. 47
2.5.9 Repeat-Accumulate code factor graph. 47
2.5.10 LDPC code factor graphs. 48
2.5.11 BTC codeword structure. 50
2.5.12 Concatenation of component decoders using equality nodes. . . . 50
2.5.13 Factor graph for an $(8,4)^2$ Hamming BTC. 51
2.5.14 Performance of some Block Turbo Codes. The BPSK limits are indicated by arrows for each code rate [37]. 53
2.5.15 Performance of various codes relative to the BPSK limit. Data points represent the SNR needed to acieve a BER of 10^{-5}. Code lengths are also indicated for regular LDPC [69], irregular LDPC [20], PCCC Turbo [15], and SCCC Turbo codes [81]. 54

3.1.1 A simple two-state convolutional code. 56
3.1.2 Valid and invalid paths in a tailbiting trellis with four sections. . . 57
3.1.3 Trellis-style constraint and factor-graph for $f(x,y,z) = x+y+z=0$. 57
3.1.4 Tailbiting trellis for the (8,4) Hamming code. 58
3.1.5 Factor graph corresponding to the tailbiting trellis of Figure 3.1.4. 58
3.2.1 Normal graph for (8,4) Hamming code. 63
3.2.2 Illustration of the "Tree" function node. 63
3.2.3 Message schematic diagram for the (8,4) Hamming decoder. . . . 64
3.2.4 Complete message and node schematic for (8,4) Hamming decoder. 65
3.2.5 Illustration of constraint between information bit and state variable (normal form). 67
3.3.1 An example row subtrellis. 68
3.3.2 Elementary subtrellises for a conventional (8,4) Hamming trellis. . 70
3.3.3 Trellis product of rows one and two. 70
3.3.4 Complete trellis after all row products are taken. 70
3.3.5 Construction of a tailbiting (8,4) Hamming code trellis. 71
3.3.6 Illustration of the squaring construction. 72
3.3.7 Synthesis of a subtrellis for the two-level squaring construction. . 74
3.3.8 A complete trellis based on the two-level squaring construction. . . 74
3.3.9 Coset subtrellis for (16,11) Hamming code. The complete trellis includes a second, identical subtrellis connected as in Figure 3.3.8. 76
3.3.10 A trellis-style graph indicating the constraints among channel bits z_i and branch label l. 77

3.3.11 Complete factor graph for the (16,11) Hamming code. 78
3.3.12 Shape of conventional trellises constructed with the two-level squaring procedure. 79
3.3.13 Shape of tailbiting trellises constructed with the two-level squaring procedure. 80
3.3.14 Tailbiting form of Figure 3.3.9. 81
3.3.15 Complete tailbiting trellis for the (16,11) Hamming code. 82
3.4.1 Tailbiting trellis for (8,4) Hamming code. 84

4.2.1 Tree representation of an infinitely large LDPC code. 92
4.3.1 Iterative behavior of $h(s,r)$. 95
4.3.2 A detailed view of $h(s,r) - r$ when s is just above, just below, and equal to the threshold. These curves are for the irregular LDPC ensemble defined in the first (leftmost) column of Table 4.2. . . . 98
4.3.3 Iterated integration using only three-edge nodes. 106
4.4.1 A sequence of estimations of $h(s,r) - r$ during a threshold search. The initial low-precision search is shown as a dashed curve. The sequence of high-precision estimates of $y_m^{(j)}$ is joined by a solid curve. 109
4.4.2 A sequence of slopes measured during threshold search. 109

5.1.1 A three-terminal translinear device. 111
5.1.2 A simple translinear loop. 113
5.1.3 Current mirror circuit. 113
5.1.4 A simple Gilbert multiplier. 114
5.1.5 A Gilbert multiplier circuit for vector scaling. 115
5.1.6 Gilbert vector multiplier. 116
5.1.7 A Gilbert-multiplier implementation of the equality node. 117
5.1.8 Differential pair circuit. 118
5.2.1 Complementary MOS devices. 119
5.2.2 Configuration of devices as current sources and diode-connected current loads. 120
5.2.3 Unsaturated weak-inversion translinear MOS device. 123
5.2.4 Canonical sum-product circuit topology. 124
5.2.5 Symbol for a source-connected transistor array. 124
5.2.6 A PMOS renormalization circuit for a two-element current vector. 127
5.2.7 Connection between two stages of NMOS sum-product circuits and PMOS normalizers. 128
5.3.1 A "slice" of the canonical topology. 129

5.4.1 Use of a reference input to restore the denominator of (5.1.12). . . 132
5.4.2 A trellis section with disjoint subsections. 134
5.4.3 A trellis section with disjoint "butterfly" subtrellises. 135
5.4.4 Implementation of a trellis section with disjoint butterfly subtrellises. Dashed lines indicate incomplete probability masses. Solid lines indicate complete probability masses. A dot indicates the row input edge for each cell. 136
5.4.5 Butterfly circuit with reference input. 137

6.1.1 Translinear loop 1, derived from Fig. 5.2.4 with $V_{\text{ref}} = 0$. 140
6.1.2 Translinear loop 2, derived from Fig. 5.2.4. 141
6.1.3 Translinear loop 3, derived from Fig. 5.2.4. 141
6.2.1 Low-voltage sum-product circuit topology, using the box notation of Fig. 5.2.5. 143
6.2.2 A "slice" of the low-voltage topology. 144
6.2.3 Allowable supply voltages for canonical and low-voltage topologies, as a function of process feature size and temperature. 148
6.2.4 Difference in minimum supply voltage between low-voltage and canonical sum-product circuits, as a function of temperature. . . . 149
6.2.5 Iterated amplification of current magnitude k. 150
6.2.6 Current magnitude transfer function for the low-voltage renormalization circuit. 152
6.3.1 A low-voltage circuit for trellis decoding based on the BCJR algorithm. 152
6.3.2 Low-voltage equality-node circuit for LDPC decoding. 154
6.3.3 Low-voltage check-node circuit for LDPC decoding. 155
6.4.1 A Tanner graph decoder for the (7,4) Hamming code. 156
6.4.2 A complete node implementation, with a separate sum-product circuit for each edge. 157
6.4.3 Simulation results showing the decoder converge to a new codeword decision. 157

7.1.1 Analog decoder interface architecture. 161
7.1.2 Buffered cascade of S/H circuits, with differential pair. 163
7.1.3 Unbuffered S/H interface circuit. 163
7.1.4 S/H circuit showing the transmission gate. 163
7.1.5 Model of leakage current in S/H circuits. 165
7.1.6 Physical origin of substrate leakage. 165

7.1.7 The increase in error probability due to comparator offsets. Offsets are assumed to be Gaussian distributed voltages, represented in the Figure in normalized log-likelihood units. 169

7.2.1 Current mirror circuit. 171

7.2.2 Density function of the mismatch factor (with exaggerated variance). 172

7.2.3 Basic Gilbert multiplier sum-product circuit. 173

7.2.4 Mismatch referral for a differential-pair circuit. V_Y represents the offset voltage needed to make $I_0 = I_1$ when the received channel sample is $r = 0$ for a particular pair of mismatch values, ε_1 and ε_2. The effective additional channel noise, n_m, is the offset in the received sample which is needed to produce V_Y. 176

7.2.5 Performance loss due to mismatch in a feed-forward circuit, for a rate-$1/2$ code at $E_b/N_0 = 1.5\,\text{dB}$. The parameter ν is varied from 0.25 to 8, as indicated to the right of each curve. 176

7.2.6 Illustration of iterated VEGAS integration for nodes of degree > 3. 177

7.2.7 The threshold loss due to mismatch for regular LDPC ensembles corrseponding to those in Table 4.3. 178

7.2.8 Comparison of feed-forward and lateral losses. 179

8.1.1 Circuit for the "Tree" computation. 183

8.1.2 Circuit implementation of $T_1(c)$. 184

8.1.3 Circuit implementation of $T_1(cc)$. 185

8.1.4 Circuit implementation of $T_2(c)$. 186

8.1.5 Circuit implementation of $T_2(cc)$. 187

8.1.6 Circuit for the T_1 (out) operation. 188

8.1.7 Circuit for the T_2 (out) operation. 189

8.2.1 Interface diagram for the (8,4) analog decoder. 190

8.2.2 Timing diagram for the analog (8,4) Hamming decoder interface. . 191

8.2.3 The input stage for the (8,4) decoder. 192

8.2.4 The unity-gain buffer circuit. 193

8.2.5 Latched current comparator circuit. 194

8.3.1 Photo of the analog (8,4) decoder chip. 195

8.3.2 Decoder performance vs speed in moderate inversion. 197

8.3.3 Measured performance in strong-inversion. 198

8.3.4 Measured performance in weak inversion, sending only one codeword. The solid curves represent uncoded BPSK and ideal Hamming code performance. Measured points are indicated by circles. Error bars indicate 99.9% confidence intervals. 200

8.3.5 Measured performance in weak inversion, varying the codeword. The solid curves represent uncoded BPSK and ideal Hamming code performance. Measured points are indicated by circles. Error bars indicate 99.9% confidence intervals. 201
8.3.6 Mixed-signal interference in outputs of the (8,4) analog decoder. . 202

9.1.1 Subdivision of the bit-combiner (Figure 3.3.10) into atomic trellis components. The labels Z_j represent information input and output for binary code variables. S represents a hidden state of the eight-state trellis graph. 204
9.1.2 Atomic subtrellises for the (16,11) Hamming code. 206
9.1.3 A block schematic for the (16,11) decoder. 206
9.1.4 Layout of the (16,11) analog decoder. 207
9.1.5 Use of equality gates at decoder outputs. 208
9.1.6 Layout of (16,11) decoder, with equality gates. 208
9.1.7 Creating a butterfly from a tree. 209
9.1.8 Upward Butterfly computation, with reference input. 210
9.1.9 Use of reset switches in the *Core* sum-product component. 211
9.2.1 Modular S/H array. 212
9.2.2 Signal timings for the bottom S/H array module. 213
9.2.3 S/H circuit for the (16,11) decoder. 214
9.2.4 Transmission gate circuit. 215
9.2.5 Latched current comparator circuit. 215
9.3.1 Photograph of the FPGA-based test board. 217
9.3.2 Screen shot of the test interface during a test of an uncoded loop-back interface chip. 218
9.3.3 Diagram of the test interface. 219
9.3.4 Results for the loop-back test at maximum speed. The solid curve indicates the ideal performance of uncoded BPSK. Circles indicate measured error rates. Error bars indicate 99.9% confidence intervals. 220
9.3.5 Loop-back results at reduced speed. The solid curve indicates the ideal performance of uncoded BPSK. Circles indicate measured error rates. Error bars indicate 99.9% confidence intervals. 221
9.4.1 Performance measurements for the (16,11) decoder. The solid curves indicate performance of uncoded BPSK and an ideal (16,11) Hamming decoder. Circles indicate measured error rates. Error bars indicate 99.9% confidence intervals. 223
9.4.2 Transient response of the (16,11) decoder output for a single bit. The solid curve indicates the output $P(\text{bit} = 0)$ and the dashed curve indactes $P(\text{bit} = 1)$. 225

10.1.1 Floorplan of the Product decoder. 229
10.1.2 Layout of the Product decoder chip. 230
10.1.3 Alternative Turbo-like interleaver layout. 230
10.2.1 Die photo of the Product decoder chip. 231
10.2.2 Test results for the Product decoder. 235
10.2.3 Error floors observed in measurements of the Product decoder. . . . 235

List of Symbols

Communication Systems

$\underline{u}$	Information message (i.e. the source bits).
$\underline{x}$	Codeword, before transmission. Often refers to the modulated codeword, as indicated by context.
$\underline{n}$	Gaussian noise pattern added by the channel.
$\underline{r}$	Received noisy samples at the channel's output.
$\hat{\underline{u}}$	Estimate of the information message, produced by the decoder.
$\underline{\rho}_i$	Conditional probability mass for codeword symbol x_i, given received sample r_i.
N_0	Channel noise power spectral density.
$g_\mu(r)$	Gaussian density function with mean μ and variance $N_0/2$.
X_i	Log-likelihood ratio for sample r_i.
E_b	Energy per information bit.
E_s	Energy per transmitted channel symbol.
SNR	Signal to noise ratio (E_b/N_0) in dB.
S	Signal to noise ratio (E_b/N_0) in unitless form.
W	Transmission bandwidth.
R	Code rate, E_s/E_b.
C	Channel Capacity (the maximum possible code rate).
$\mathcal{B}(r)$	The BPSK Limit (more restrictive than Capacity).

Block Codes

$\mathcal{C}$	Block code.
k	Length of the uncoded information block.
n	Length of a codeword.
G	Generator matrix.
H	Parity-check matrix.
$\mathbb{Z}_2$	Finite field of integers modulo 2.
$\mathbb{Z}_2^k$	Space of length-k vectors over $\mathbb{Z}_2$.
$\underline{x}_0$	The all-zero codeword.
$h(\underline{x}_1,\underline{x}_2)$	Hamming distance between $\underline{x}_1$ and $\underline{x}_2$.
$d_{\min}$	Minimum Hamming distance between pairs of codewords in $\mathcal{C}$.
P_e	The bit error probability for $\mathcal{C}$, for a given set of channel parameters.
$A(w)$	The multiplicity of codewords of weight w in $\mathcal{C}$.

Factor Graphs

$f(X)$	A factorable function of several variables.
X	The set of variables on which f depends.

$f_j(X_j)$	A factor of $f(X)$.
X_j	A subset of X.
$\mathcal{G}$	A factor graph.
$\mathcal{T}(H)$	The Tanner graph induced by parity-check matrix H.
$H'(\mathcal{T})$	The parity-check matrix induced by Tanner graph $\mathcal{T}$.
$\mathcal{N}$	A normal graph.

Sum-Product Algorithm

$\mathbf{x}$	A discrete-type random variable (distinguished by bold-face type).
$\mathcal{A}_{\mathbf{x}}$	The set of possible values for $\mathbf{x}$.
$\underline{\rho}_{\mathbf{x}}$	The probability mass for $\mathbf{x}$.
$\underline{\rho}_{\mathbf{x}}(i)$	The i^{th} component of $\underline{\rho}_{\mathbf{x}}$.
$f(x,y,z)$	A boolean constraint function on three discrete-type variables.
S_f	The set of (x,y,z) for which $f(x,y,z)$ is satisfied.
$S_f(j)$	Subset of S_f for which $z = j$.
N	A three-edge function node.
E	An edge in the graph.
η	A normalizing constant.

Iterative Decoders

C_1, C_2	Constituent codes.
Π	Interleaver which permutes bit order between C_1 and C_2.
$\underline{u}$	Information bits.
$\underline{p}_1$	Parity bits from C_1.
$\underline{p}_2$	Parity bits from C_2.

Trellis Codes

$\mathcal{S}_i$	A column of trellis states.
$\mathcal{L}_i$	The set of trellis branches with $\mathcal{S}_i$ on the left.
$\mathcal{T}_i$	A trellis section.
s_q	A left-hand state in a trellis section.
s'_r	A right-hand state in a trellis section.
b_{qr}	The branch connecting s_q on the left to s'_r on the right.
l_{qr}	The label on branch b_{qr}, consisting of a channel symbol and, optionally, a corresponding information symbol.
$\mathcal{A}_s^{(i)}$	The set of possible states at time i.
λ_i	Probability mass for channel symbol x_i, locally conditioned on the received sample r_i.
ρ_i	Probability mass for the information symbol u_i, globally conditioned on the received sample block, $\underline{r}$.

σ_i	Probability mass for $\mathcal{S}_i$, globally conditioned.
γ_{jk}	State transition probability; the probability of moving to state k on the right, given that we are in state j on the left.
Γ_i	Transition probability matrix for $\mathcal{T}_i$.
$\mathcal{T}_i \otimes \mathcal{T}_{i+1}$	Trellis section product, merging $\mathcal{T}_i$ with $\mathcal{T}_{i+1}$.
$\underline{\alpha}_i$	Forward-propagating (backward-conditioned) state probability mass in the BCJR algorithm.
$\underline{\beta}_i$	Backward-propagating (forward-conditioned) state probability mass.
$\underline{\alpha} \odot \underline{\beta}$	The term-by-term product of $\underline{\alpha}$ and $\underline{\beta}$.
$\mathcal{D}(\underline{a})$	Decision operation, returning the index of the maximum element in vector $\underline{a}$.

Block Code Trellises

g_i	A row of block generator matrix G.
$\mathcal{R}_i$	Elementary subtrellis for g_i.
$\mathcal{R}_1 \odot \mathcal{R}_2$	Product of subtrellises, creating a larger subtrellis.
$l_1 \oplus l_2$	Group product of branch labels.
a_i	Number of active rows in column i of G.
$\lvert\mathcal{S}_i\rvert$	Number of states in $\mathcal{S}_i$.
L	Length of trellis (number of sections).
$\mathrm{RM}(\rho, \nu)$	Reed-Muller code with parameters ρ and ν.
T/V	A partition of set T induced by coset V.
$\lvert T/V\rvert^2$	Squaring operation applied to the partition T/V.
$T/U/V$	Two-level partition of set T, induced by the partition U/V.
$\lvert T/U/V\rvert^4$	Two-level squaring operation.

LDPC Codes

d_c	Check node degree (i.e. row weight in the parity check matrix) for regular codes.
d_v	Variable node degree (column weight).
$C^n(d_v, d_c)$	Ensemble of (d_v, d_c)-regular LDPC codes of length n.
$\Pi(i)$	Interleaving function mapping variable node connections to check node connections.
r	Design rate of an LDPC ensemble,
$\gamma(x)$	An edge-oriented degree distribution polynomial for an irregular LDPC code.
λ	Variable-node degree distribution (edge-oriented).
ρ	Check-node degree distribution (edge-oriented).
$\int\gamma$	Average inverse degree (serves as a normalizing constant).
$\tilde{\lambda}$	Node-oriented degree distribution.
$\mathcal{N}_{\mathcal{E}}$	Total number of edges in the code's Tanner graph.

(λ,ρ)	Parametric description of an irregular LDPC code ensemble.

Density Evolution

s	Channel parameter, defined as $E_s/N_0 = 1/N_0$.
s^*	The threshold (best possible s) for a given LDPC ensemble.
Y	A random LLR message.
σ_Y^2	The variance of Y.
m_Y	The mean of Y.
$Q(x)$	Gaussian error integral, from x to ∞.
$r^{(l)}$	Error probability at variable nodes' output, after l iterations.
$h(s,r)$	Iterative transfer function for $r^{(l)}$.
$\phi()$	Function which converts mean LLR to mean probability mass.
α,β,γ	Scalar parameters used for approximating ϕ.

Numerical Methods for Density Evolution

U	Set of i.i.d. random LLR messages from check nodes, arriving at a variable node.
V	Set of i.i.d. messages from variable nodes, arriving at a check node.
$f_c(V)$	Check node function, governed by the sum-product algorithm.
$f_v(U)$	Variable node function.
ρ_U	PDF of an element of U.
μ_U	Mean of an element of U.
ε_i	Estimation error resulting from a Monte Carlo integration.
μ_i, σ_i^2	Mean and variance of ε_i.
y	Estimate of $h(s,r)-r$.
Δr	Fixed distance between points for measuring $\partial y/\partial r$.
y_m	The maximum value of $y(r)$ for a given s.
r_m	The r for which $y = y_m$.
α	Parabolic width parameter used for approximating $y(r)$ near r_m.
ε_r	Error in estimate of r_m.
$\varepsilon_{\text{slope}}$	Error in estimate of $\partial y/\partial r$.
ε_y	Error in estimate of y.
ε_m	Error in estimate of y_m.
ε_s	Error in estimate of threshold, s^*.
β	The slope of y_m w.r.t. s at $s = s^*$.

Iterated Monte Carlo Integration

$\mathcal{M}_u(\mu_v)$	Monte Carlo mean-output estimator for check nodes, with mean LLR input μ_v.
$\mathcal{M}_v(\mu_u)$	Monte Carlo mean-output estimator for variable nodes.

μ_u	The result of the estimator $\mathcal{M}_u$.
ε_u	The error in the estimate μ_u.
μ_{out}	Mean LLR output by variable nodes.
σ^2_{out}	Variance of the estimation of μ_{out}, accounting for the estimation variance of μ_u.
χ	The slope of μ_{out} w.r.t. μ_{in}, evaluated at μ_{in}, for a particular Monte Carlo estimation.
$\rho(\mu,x)$	Gaussian LLR density function with mean μ.

MOS Transistor Parameters

I_D	Current through an MOS device, flowing from drain to source.
V_{th}	Threshold voltage, at which the device turns on.
V_{T0}	Threshold voltage with zero body effect.
μ	Mobility of charge carriers in the channel (may be N-type or P-type carriers).
C'_{ox}	Gate oxide capacitance per unit area.
W	Width of the physical transistor gate.
L	Length of the gate.
κ	Subthreshold channel pinch-off voltage slope (always less than 1).
U_T	The thermal voltage k_BT/q.
I_S	Specific current, designating the boundary between weak and strong inversion.
$f_s(v_s)$	Function describing small non-linear modulation of device current due to v_s.
$f_g(v_g)$	Non-linear device current effect due to v_g.
$f_{ds}(v_{ds})$	Non-linear effect due to v_{ds}.
v_{gs}	Voltage between the gate and source of the MOS transistor.
v_{ds}	Voltage between the drain and source.
v_g	Voltage applied to the gate.
v_d	Voltage at the drain.
v_s	Voltage at the source.
g_m	Transconductance, the slope of I_D w.r.t. v_{gs} about a fixed bias current I_{D0}.

Analog Sum-Product Circuits

Ix_i	A row input (current) to a Gilbert multiplier or sum-product circuit.
x_i	The node (voltage) to which Ix_i is connected.
Iy_j	A column input to a sum-product circuit.
y_j	The node to which Iy_j is attached.
Iz_{ij}	An output from a Gilbert multiplier or sum-product circuit.
$\mathbf{x}$	A discrete-type random variable.
$\mathcal{A}_{\mathbf{x}}$	The set of possible values for $\mathbf{x}$

I_U	Unit current, representing a probability of one.
A'	Normalized current unit; $1A' = I_U$.
V_X	A differential voltage proportional to LLR X.
V_X^+, V_X^-	The positive and negative components of V_X.
X	Ordered sequence of row input nodes.
Y	Ordered sequence of column input nodes.
N	Number of row inputs.
M	Number of column inputs.
k_Z	Current magnitude; the sum over all elements of a current vector, in normalized units A'.
n, m	Geometry width factors for MOS devices in a renormalization circuit.
V_{ref}	Reference voltage applied to the source of row-input transistors, to ensure that column-input transistors remain in saturation.

Reference Input Simplification

δX	A subset of row-inputs.
Δ	A reference or remainder input, representing the sum of currents in $\mathrm{X} \setminus \delta X$.
δN	The number of elements in δX.
δY	A subset of column inputs.
δM	The number of elements in δY.
$\mathrm{X} \otimes \mathrm{Y}$	The set of all pairwise products between members of set X and those of set Y.

Low-Voltage Sum-Product Circuits

Id_i	Cumulative source current in column i.
I_δ	Cumulative current through dummy transistors in a given column.

Analog Decoder Interfaces

s	Scaling parameter relating LLRs to differential voltages, with units V/LLR.
I_{leak}	Substrate leakage current in transmission gates.
V_C	Voltage stored on a S/H capacitor C.
$V_{\max}$	The maximum allowed drop in the common-mode of a stored differential voltage.
$t_{\max}$	The time required for I_{leak} to cause a loss equal to $V_{\max}$.
τ_s, f_s	Sample time and frequency for a S/H circuit.
R_{on}	Series resistance in a transmission gate.

N_S	Maximum number of samples which can be serially stored for a given block length and sample rate, before $t_{\max}$ is exceeded in at least one S/H cell.
ΔQ	Extra charge deposited on C due to charge-injection by the transmission gate.
k	Linear slope of ΔQ w.r.t. V_C.
X_o	The comparator's input offset, expressed as a LLR.
L	Maximum tolerated value of $\lvert X_o \rvert$.
ε	Mismatch error term which modulates a device's current by a random factor.
σ_n^2	Channel noise variance, $\sigma_n^2 = N_0/2$.
σ_m^2	Variance of the mismatch term ε.
σ_{tot}^2	Total effective noise at the decoder's input (i.e. at the channel's output).
ν	Topology-dependent mismatch factor which can increase or decrease the effective mismatch variance.
$\mathcal{N}$	Analog sum-product node.
X, Y	Input LLR messages to $\mathcal{N}$.
μ_X, μ_Y	Mean of X and Y.
$f_n\left(X, Y, \overrightarrow{\varepsilon}\right)$	Function characterizing the behavior of $\mathcal{N}$.
Z	The output LLR message from $\mathcal{N}$.
μ_Z	The mean of output message Z.
$s^*\left(\sigma_\varepsilon\right)$	The threshold of an LDPC ensemble, given mismatch standard deviation σ_ε.
s^*_{exact}	The ideal threshold of the same ensemble.
$f_{\text{loss}}\left(\sigma_\varepsilon\right)$	The ratio of the mismatch threshold to the ideal threshold. When expressed in dB, this is an estimate of the SNR loss caused by lateral mismatch.

List of Abbreviations

ADC	Analog to digital converter.
AMI	AMI Semiconductor Corporation.
AWGN	Additive white Gaussian noise.
BER	Bit error rate.
BCJR	Bahl, Cocke, Jelinek, Raviv (the authors who created the algorithm).
BPSK	Binary phase-shift keying.
CDROM	Compact disc read-only memory.
BTC	Block Turbo Code.
CMOS	Complementary MOS technology.
DAC	Digital to analog converter.
FIR	Finite-impulse response.
GA	Gaussian approximation.
GNU	GNU is Not Unix – the GNU project.
i.i.d.	Independent and identically distributed.
LDPC	Low-density parity check (code).
LLR	Log-likelihood ratio.
MAP	Maximum a-posteriori.
ML	Maximum likelihood.
MOS	Metal-oxide-semiconductor device.
NMOS	Negative MOS device.
PCCC	Parallel concatenated convolutional codes.
PMOS	Positive MOS device.
PN	Positive-negative junction diode.
RM	Reed-Muller code.
RV	Random Variable.
SCCC	Serially concatenated convolutional codes.
S/H	Sample and hold.
SNR	Signal to noise ratio.
SP	Sum-product algorithm.
TPC	Turbo Product Code.
TSMC	Taiwan Semiconductor Manufacturing Corporation.
UMTS	Universal Mobile Telecommunications System.
w.r.t.	With respect to.

Chapter 1

Introduction

Error control codes are used in a variety of applications to reduce or effectively eliminate the occurrence of errors in information which is transmitted or stored. The *Shannon Capacity Limit* has long been known as a bound on the performance of error control systems [78, 23]. The fundamental problem of error control coding has been to produce a system which achieves this limit. In 1993, with the introduction of *Turbo Codes*, this problem was effectively solved for the additive Gaussian noise channel [15, 16]. It was quickly discovered that several other types of known codes – such as Low Density Parity Check codes [33] and Block Product codes [28] – could also approach Capacity on the Gaussian channel [56, 69, 37, 67].

An important part of the Capacity problem still exists: to implement an efficient error control coding system which not only approaches capacity, but does so at a minimal expense. Turbo codes achieve their good performance through *iterated estimation* of the transmitted message. First, an entire block of data is received and decoded. Then the same block is decoded again and again, with improved results after each iteration.

All of these iterations must occur at a much higher rate than that of the data itself. This means that the decoder must operate at a very high clock frequency if large data rates are desired. This results in a substantial power drain, and generates a significant amount of heat. It may also cause high-frequency interference in nearby circuits.

An alternative to a high-speed decoder is a fully parallel decoder. Iterative decoding algorithms are naturally parallel and distributed. A fully parallel decoder only needs to operate at a speed proportional to the data's *block rate* (or *frame rate*). If the size of a frame is very large, then a correspondingly high *bit rate* is achievable.

In a parallel architecture, decoding iterations can also be *pipelined* through many instances of the decoder circuit [18]. Decoding iterations are thereby distributed across space rather than time, leading to very high-speed decoding. This approach also has a high cost in power consumption, as a system can be extremely large. An even greater cost lies in the circuit area required for a large parallel implementation. The cost of manufacturing a semiconductor chip is directly proportional to area. When yield is taken into account in bulk semiconductor production, cost is proportional to the area cubed [68].

The benefit of an error control system is that it allows error-free transmission with reduced signal energies, or extended range, or increased density in storage media. Iterative decoders run the risk of overshadowing this benefit with the high cost of the error control system itself. System designers are therefore faced with task of assessing the *total system efficiency* of any error control code implementation. There is still a great deal of argument about how to best approach this problem.

Frustrated with the cost and inefficiency of iterative decoder implementations, some researchers have begun to explore *analog computation*. Very simple analog circuits implement the basic operations needed for iterative decoding [38, 52]. The physical implementations of these circuits are much smaller than their digital counterparts, and typically require at least an order of magnitude less power.

Analog decoders therefore provide a means of improving the power-efficiency and circuit complexity of iterative decoders. Such an improvement would make capacity-approaching error control codes suitable for a wider variety of applications. While analog computational circuits are known for their imperfections, these imperfections are expected to be corrected by the error-reducing nature of iterative decoders.

Many problems which arise in analog signal processing are reduced in analog decoders because their circuit structure is *fully differential*. Information is never communicated on a single wire. Instead, information is encoded in the *relative* *proportions* of two or more analog currents. Many instances of signal interference or device imperfection tend to affect all components of differential signals equally. While the absolute values of each signal are altered, the relative values remain intact.

Several groups have announced successful results implementing small analog decoders [54, 59, 86, 35]. Most recently, an analog Turbo code has been announced which implements a code defined in the UMTS communication standard [45]. This is the first analog decoder to a complexity sufficient to meet the requirements of a commercial standard. While this most recent development is quite significant, no analog decoder has yet been large enough to provide capacity-approaching performance. The successful chips produced so far nevertheless demonstrate the concept of analog decoding, and provide estimates of their performance when compared against digital decoding circuits.

Some initial comparisons of analog and digital decoders are shown in Figures 1.0.1 and 1.0.2. The "performance" in these graphs is defined as the *signal-to-noise ratio* (in dB) required to achieve a *bit error-rate* of 10^{-3} errors per bit. While based on only a few samples of early designs, these figures illustrate the trends which are expected to hold between analog and digital decoding circuits. Figure 1.0.1 shows the improvement in circuit size offered by analog decoding circuits over digital circuits, when implemented in the same technology. Similarly, Figure 1.0.2 shows the improvement in power consumption offered by analog decoders. All decoders in these figures are normalized to a 0.5μm process, and are assumed to operate at 2Mbits per second.

In these figures, there are two labeled data points which represent analog decoders. The (8,4) Hamming analog decoder represents measured results from a design presented in Chapter 8 of this thesis. The $(16,11)^2$ Product Decoder represents the results of a design presented in Chapter 10 of this thesis. The product decoder data is adjusted to account for differences in the semiconductor process. The figures represent a process with a supply voltage of 3.3V and a minimum feature size of 0.5μm. The product decoder is implemented in a 1.8V, 0.18μm process. The actual size and process-adjusted size are indicated explicitly in Figure 1.0.1.

The remaining data points represent digital decoders. The data points are derived from a study on size and power consumption of various digital decoders published by Worthen et. al. in 1999 [94]. The data include a range of decoders, representing Turbo decoders, convolutional (Viterbi) decoders, and the simplest hard decision decoders. All decoders in this study were synthesized and simulated, but not actually fabricated. This data set is especially useful for comparisons because all digital decoders were synthesized for the same 0.5μm process using the same methodology.

The actual improvement of analog over digital decoders is probably larger than shown in Figures 1.0.1 and 1.0.2 because interface and storage circuits are accounted for in the analog circuits, but not in the digital ones. A digital decoder also requires a digital-to-analog converter at its input, which significantly increases its area and power consumption. The plots also do not account for differences between codes such as code rate, bandwidth efficiency, or data block size.

1.1 History of analog decoding.

Analog decoding circuits have a long history in communications research. The earliest analog soft-information decoders were analog implementations of the Viterbi algorithm. Analog Viterbi decoders were reported at least as early as the mid-70's [4]. The Viterbi algorithm is a method of maximum-likelihood decoding for trellis codes, and is asymptotically optimal on Gaussian noise channels [29, 73].

Viterbi decoders implement a relatively simple sequential algorithm which is useful for a wide range of applications, including equalization of magnetic recording channels, decoding for satellite channels, and wireless communication channels. Because of these features, the Viterbi algorithm is very widely used. Interest in analog Viterbi decoders has remained steady over the years, and numerous authors have reported analog circuits which vastly outperform digital solutions in terms of speed and/or power and/or complexity [26, 47, 57, 77, 76, 80, 83].

The field of analog *iterative* decoders began with nearly simultaneous proposals by Loeliger, Lustenberger et. al. [54] and by Hagenauer, Moerz et. al. [38]. These

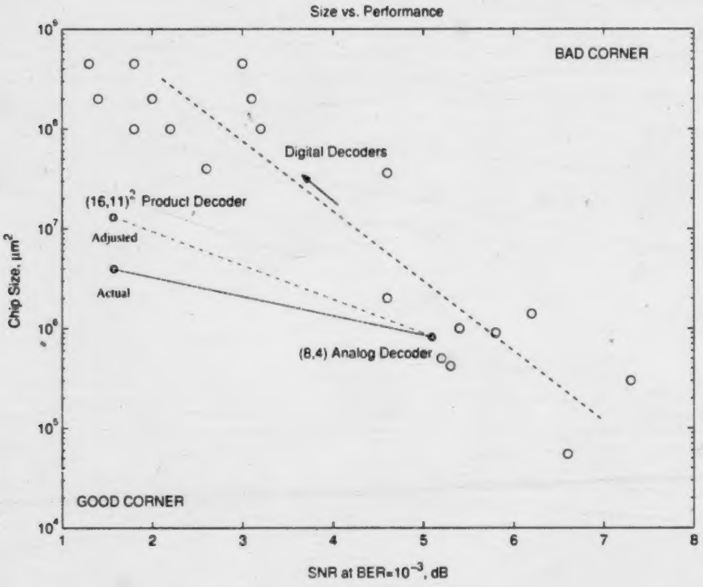


Figure 1.0.1: Decoder size vs performance, comparing a collection of synthesized digital decoders in a 0.5μm CMOS process [94]. Two analog decoders are shown as labeled. The Product decoder is shown adjusted for the difference in process feature size. The analog decoders are reported in this thesis.

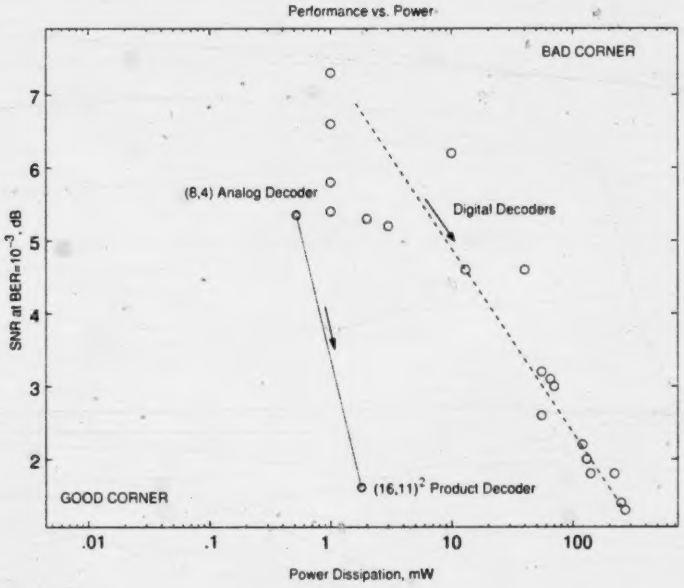


Figure 1.0.2: Decoder power vs performance, showing a collection of simulated digital decoders [94] alongside two fabricated analog decoders. The analog decoders are labeled as an (8,4) Hamming decoder and a $(16,11)^2$ Product decoder. Results for the analog decoders are reported in this thesis.

authors proposed using non-linear analog circuits, such as those used in neuromorphic analog circuits [58, 51], to implement decoding operations.

Analog iterative decoders thus emerged as a means of implementing Turbo codes and LDPC codes. The standard Viterbi algorithm is insufficient for these codes, because the decoder's output is binary (i.e. *hard* information). A more complex algorithm, known as BCJR [9], is required for Turbo codes. The output of a BCJR decoder is *soft*, allowing it to be exchanged with a network of decoders for multiple rounds of decoding.

In 2001, Kschischang, Loeliger and Frey consolidated the theory of soft-information algorithms, including BCJR. These algorithms were shown to be instances of a general algorithm, called the *sum-product algorithm*, which is described in terms of a type of graph called the *factor graph* [48]. It was further shown by Loeliger and Lustenberger that a large class of factor graphs could be mapped to analog circuits [52]. At the same time, Forney introduced a refined factor graph notation, called the normal graph, which clarified the correspondence between a code, its graph, and the sum-product implementation [31]. In 2000, Lustenberger demonstrated that the normal graph approach provides a systematic methodology for the synthesis of analog decoding circuits [53].

The principles of analog iterative decoding were also hinted at by earlier work. As early as 1979, Rudolph et. al. proposed an analog soft algebraic decoder based on sum and product operations [71]. This proposal was concerned with implementing an asymptotically optimal soft decoding algorithm called "maximum radius" decoding. Rudolph's decoding algorithm applies specifically to cyclic block codes, which are widely used but are far less powerful and complex than Turbo codes.

Rudolph's algorithm appears very similar to the sum-product algorithm when implemented on a systematically simplified factor graph. The proposed design even included "iterative extensions" in which the decoder's soft output is fed back to its input. It is now known that such an iterative arrangement cannot achieve capacity-approaching performance. Rudolph's design was, however, painfully close to the current structure of an analog iterative decoder. It was clearly a work ahead of its time.

The idea of applying analog neuromorphic circuits to analog decoding also arose in earlier research. The earliest occurrence was evidently in 1996, when Wang and Hacking described an "artificial neural net Viterbi decoder" [83]. Their proposal was to implement a parallel network of analog "neurons" for Viterbi decoding. By computing in parallel, a higher speed is achieved. They also argued that the analog decoder would have better power consumption and complexity than a digital implementation. Precisely the same arguments are made in favor of analog iterative decoders. Neuromorphic analog Viterbi decoders were revisited by Kim et. al. [47] in 2004.

1.1.1 Physical demonstrations.

Early demonstrations (1999-2000).

Physical demonstrations of analog decoders were produced as early as 1999. Lustenberger, Loeliger et. al., of ETH Zurich, were the first to publish results for an actual chip [54], and were quickly followed by Moerz, Hagenauer et. al. [59], of the Munich University of Technology (TUM). These decoders were implemented in BiCMOS processes, and used bipolar junction transistors to perform the decoding operations. In both cases, the authors also proposed using subthreshold CMOS circuits to implement the decoding operations, but chose not to implement them initially. A purely CMOS design, they argued, is desirable from the perspectives of power, cost, and integrability.

In 2000, a fully-CMOS analog decoder chip was designed by the author of this thesis, in collaboration with researchers at the University of Utah. Results for this design were published in early 2001 [86]. This chip demonstrated micropower analog decoding in a standard CMOS process.

A larger BiCMOS analog decoder was attempted by Lustenberger, as reported in 2000 [53]. This design was evidently not successful. It employed digital-to-analog conversion circuits at the input. Digital inputs were chosen to simplify testing. It seems likely that the digital-to-analog converters failed to function properly, resulting in the design's overall failure.

While the early chips demonstrated the principle and the possibilities of analog decoding, they all had several notable flaws. Most severe among the flaws was the lack of an appropriate interface. All three chips used fully parallel interfaces, making testing difficult. Furthermore, real communication channels tend to send data in serial. It needed to be demonstrated that analog channel samples could be reliably stored without reversing the advantages of analog decoders.

In 2001, a second analog decoder chip was designed by the author of this thesis, in collaboration with researchers at the University of Utah. This chip incorporated an analog serial-to-parallel converter based on an array of sample-and-hold capacitors [87]. The chip was fabricated in a standard CMOS process, and test results were published in January 2004 [90]. The second Utah decoder was the first to demonstrate a small, micropower, fully CMOS analog decoder with a practicable interface.

More complex designs (2001-2004).

The first crop of analog decoder chips implemented simple codes of limited value. By 2002, several groups were working toward more powerful demonstrations. Xotta, Amat, et. al., in collaboration with researchers at Torino, Padova, and ST Microelectronics, proposed the design of an analog Turbo decoder [95]. Their initial proposal was to implement a Turbo code for magnetic recording channels, with a length of about 500. This design proved overly ambitious at that time.

Amat et. al. revised their proposed design in 2003, opting to implement a UMTS standard Turbo code with a coded length of 120. This design was fabricated, and their results were published in 2004 [45], making it (quite temporarily) the largest and best performing analog decoder.

The design was based on the methodology of Lustenberger [53], and used a serial interface very similar to the Utah decoder [90]. A unique feature of their design is a single, successful digital-to-analog converter at the decoder's input. The chip's inputs were therefore digital, but the channel samples were stored internally as analog voltages. Fully digital input and output allows this chip to be used as a drop-in replacement for standard digital decoders.

The first analog Turbo decoder was realized by Gaudet, et. al. in 2002 [34].

Complete results were published in 2003 [35]. Gaudet's Turbo code had a coded length of 48, and was at that time the largest and best performing analog decoder. Gaudet's decoder introduced a programmable analog interleaver, making it the first reconfigurable analog decoder. The design also made use of novel decoding circuits which differed from the Loeliger and Hagenauer approaches. The chip used a multichannel analog serial-to-parallel conversion at its input. Each input channel was processed through a sample-and-hold array, much like the Utah design. Use of multiple serial input channels made it possible for the decoder to operate at higher speeds.

In 2003, Perenzoni et. al. (who were affiliated with Amat et. al.) reported a CMOS analog Block Turbo decoder (a.k.a. Turbo Product decoder) [66]. This code was represented as an LDPC-style code, and had a coded length of 40. It was, at that time, the largest CMOS iterative analog decoder based on Loeliger and Lustenberger's methodology. The design included a serial analog input interface with variable-gain amplifiers (VGAs). The VGAs allowed on-chip tuning of the channel noise parameter. In other interface designs (including those reported in this thesis), the input scaling done off-chip. To the author's knowledge, physical test results for Perenzoni's decoder have not been published as of the writing of this thesis.

An analog Turbo Product decoder was proposed in 2001 by the author of this thesis, in collaboration with researchers at the University of Utah [85]. The proposed design was for a length-256 analog decoder. The implementation of this decoder is the chief accomplishment reported in this thesis. The Product decoder effort was moved in 2002 to the University of Alberta. The design was finally completed in 2003, and the physical chip returned from fabrication in 2004. Based on performance measurements, the chip is deemed a success, making it the largest and best performing analog decoder to date. It is also the only analog decoder to demonstrate performance *superior* to known digital designs.

Novel circuits and technologies (2002-2004).

In 2002, Moerz et. al. sought to demonstrate a very high speed analog decoder using a silicon-germanium technology [36]. This decoder also implemented a small, weak code (the very same code implemented in their earlier designs). At present, it is not known to the author whether this design was a success. Further results have evidently not been published, and Moerz's doctoral dissertation is still forthcoming as of the writing of this thesis.

A high speed silicon-germanium analog Turbo decoder was attempted in 2003 by Huang et. al. of the University of Virginia [43, 44]. This design was evidently not successful. It employed digital-to-analog converters at the input. As with Lustenberger's failed design, the digital-to-analog converters are suspected to be the root of this chip's failure. To the author's knowledge, there has been no report of a successful silicon-germanium analog decoder as of the writing of this thesis.

A low-voltage analog decoder was designed in 2004 by Nguyen, in collaboration with the author of this thesis and others at the University of Alberta [64]. The low-voltage design implemented a small, weak code, but demonstrated the concept of analog decoding with a power supply of less than one volt. The low-voltage decoder requires the lowest energy-per-bit of any known CMOS soft-information decoder.

Digital implementations (1995-2004).

To provide context for the analog decoder achievements described above, it is important to compare them directly against digital implementations. While Turbo codes are becoming widespread in communications systems, details have been published for relatively few integrated iterative decoder chips.

Of the published digital implementations, the earliest was announced by Berrou et. al. in 1995 [14]. Several other chips have since been reported, with steady improvements in energy efficiency (measured in Joules per information bit) and chip size. Results for several digital decoders are summarized in Table 1.1. In this table, the coding gain refers to the improvement in signal-to-noise ratio (SNR) relative to uncoded BPSK modulation, at a bit error rate (BER) of 10^{-3}. This somewhat high

BER is chosen out of fairness to some decoders which have relatively high error floors.

For comparison, a summary of fabricated analog decoders is shown in Table 1.2. In many cases, the speed of a decoder is estimated using less-than-rigorous arguments based on decoder dynamics. Some of the results of this thesis, reported in Section 8.3.1, add experimental support to these estimates.

A plot comparing the energy efficiency of analog and digital decoders is displayed in Figure 1.1.1. The analog decoders mostly have better energy efficiency by up to an order of magnitude.

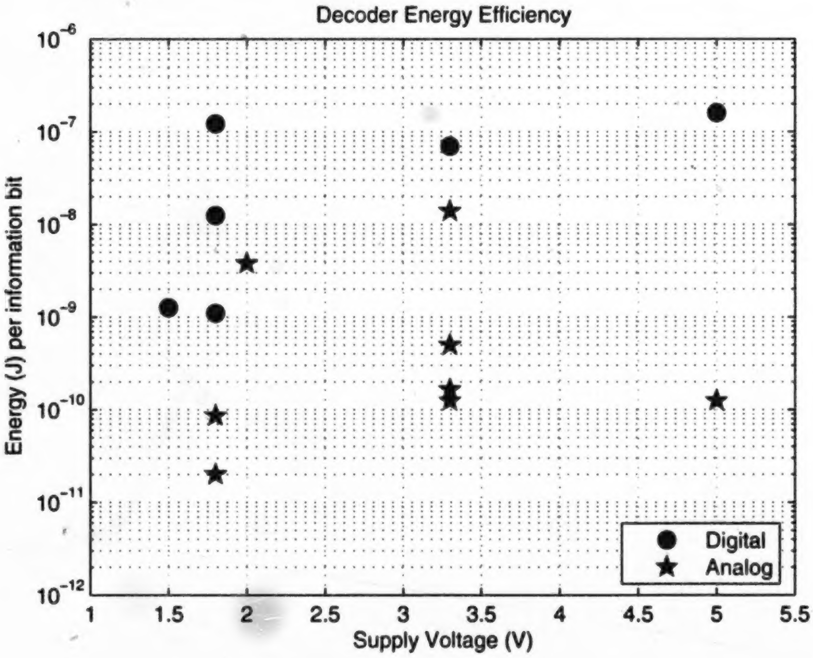


Figure 1.1.1: Energy expense per bit for analog vs digital decoders. The circles represent reported digital decoders, and the stars represent fabricated analog decoders.

1.1.2 Development of design principles.

Synthesis and simulation (2001-2004).

In his doctoral thesis, Lustenberger constructed a systematic methodology for the design of analog sum-product circuits, and for the synthesis of complete decoders.

Table 1.1: A summary of published digital iterative decoder designs.

Reference	Type	Uncoded length	Speed	Technology and Supply Voltage	Power	Size	Coding gain	J/bit
Berrou '95 [14]	PCCC Turbo	2048	40M	0.8μm 5V CMOS	1.6W per it.	-	6.3dB	160n
Hong '98 [42]*	PCCC Turbo	256	1M	0.6μm 3.3V	70mW (est.)	10.25mm × 13.65mm	3.8dB	70n
Bickerstaff '02 [17]	3GPP Turbo	5114 max	2.5M (10 it.)	0.18μm 6M 1.8V CMOS	306mW	9mm^2	6.3dB max	122n
Blanksby '02 [18]	LDPC	512	500M	0.16μm 5M 1.5V CMOS	630mW	7.5mm × 7.0mm	4.3dB	1.26n
Kaza '04 [46]*	3GPP Turbo	2048	6M	0.13μm 1.8V CMOS	6.63mW	1.07mm^2	5.3dB	1.1n
Al-Mohandes '04 [6]*	3GPP Turbo	5114 max	5M	0.18μm 1.8V CMOS	63mW	0.6mm^2**	N/A	12.5n

*** Results are for a synthesized design, evidently not fabricated.**

**** This area estimate does not account for memory.**

Table 1.2: A summary of published analog iterative decoder designs.

Reference	Type	Uncoded length	Speed	Technology and Supply Voltage	Power	Size	Coding gain	J/bit
Lustenberger '99 [54]	Tailbiting BCJR	9	100M	0.8μm 2M 5V BiCMOS	50mW	1.7mm × 0.7mm	N/A	125p
Moerz '00 [59]	Tailbiting BCJR	8	160M (est. max)	0.25μm 3.3V BiCMOS	20mW	1.68mm^2	1.6dB	125p
Winstead '00 [86]	Tailbiting BCJR	4	20M (est. max)	0.5μm 3M 3.3V CMOS	3.3mW	2.25mm^2(total die size)	1.5dB	165p
Gaudet '03 [35]	PCCC Turbo	16	13.3M	0.35μm 3.3V CMOS	185mW	1.1mm × 1.26mm	1.9dB	13.9n
Amat '04 [45]	3GPP Turbo	40	2M	0.35μm 2V CMOS	7.6mW	3.7mm×1.1mm	3.6dB	3.8n
This work, Chapter 8	Tailbiting BCJR	4	2M	0.5μm 3M 3.3V CMOS	1mW	0.083mm^2 (core area)	1.6dB	500p
This work, Chapter 9	Tailbiting BCJR	11	135M (est. max)	0.18μm 6M 1.8V CMOS	2.69mW	0.0266mm^2 (core area)	2.2dB	20p
This work, Chapter 10	Turbo Product	121	1G (est. max)	0.18μm 6M 1.8V CMOS	86.1mW	2.85mm^2	5.3dB	86p

Significant attention was given to the automatic computer-aided design of analog decoders [53]. Lustenberger also explored the requirements of high-speed analog interfaces, and the sensitivity of analog decoders to device mismatch [55].

In 2001, Jie Dai et. al. of the University of Utah reported additional progress in the area of automatic synthesis [25]. The Utah synthesis methodology added a new type of sum-product building block, called the "reference cell" (described in Section 5.4 of this thesis). Jie Dai constructed a library of cells consisting of all possible cell structures below a certain complexity. He also produced software to translate a code's factor graph description into a cell-based decoder circuit [24].

Jie Dai's software was also capable of translating the cell-based decoder description into a VHDL model of the analog decoder. The VHDL model incorporated single-pole dynamic models of the cells, allowing high-level analog hardware simulation for analog decoders. Based on simulations with these models, Jie Dai predicted that a reset circuit – a circuit which initializes the decoder's state before decoding – is necessary for proper operation. Experimental evidence for this conclusion has only recently emerged (see Section 8.3.1 of this thesis). The idea of the reset circuit was not new – it was originally suggested by Lustenberger – but Jie Dai's simulations provided concrete justification for using it.

The VHDL approach of Jie Dai was continued by Xotta and Amat et. al., who used similar one-pole models to simulate their analog Turbo decoders [45, 95]. Their results were not unlike Jie Dai's. A reset circuit was included in their final design, although it malfunctioned. The performance of their decoder was somewhat worse than expected. The poor results are blamed on the lack of a functioning reset circuit.

Through the work of Lustenberger, the design of analog sum-product circuits seems to be well-established. In this thesis, the Lustenberger methodology is described as *canonical*. The canonical approach has been validated by high-level simulations as well as physical demonstrations.

In 2003, the author of this thesis, in collaboration with Christian Schlegel at the University of Alberta, applied the method of *importance sampling* to physical-level SPICE simulations of analog decoders [88]. These simulations validated the performance of canonical analog decoders in various physical processes, under the influence of sub-micron transistor effects. Simulations were also performed to study the effect of overbiasing canonical analog decoders. This study strengthened the belief that analog decoders should be robust to a broad range of defects.

Device mismatch (2001-2004).

Researchers in analog decoding have long speculated that analog decoder should be robust against device mismatch. Mismatch is a fatal phenomenon for many analog solutions. Its effects are not directly captured by SPICE simulations, and can be difficult to predict. Writing in 2001, Lustenberger treated mismatch as the analog equivalent of quantization noise [55].

Lustenberger and Jie Dai separately presented results from large batches of analog decoder simulations, accounting for mismatch [55, 53, 24]. The results showed an undeniable resilience of the decoders under various degrees of mismatch. In 2003, Frey, Merkli and Loeliger conducted experiments using a set of discrete "soft gate" chips [32]. These chips had inter-chip mismatch on the order of 10%. They found that very little performance loss could be attributed to the mismatch. While these experiments added weight to the mismatch-robustness theory, the evidence was still only anecdotal. It remained unclear whether analog decoders would still be robust when applied to very large, complex codes.

The first analytical study of mismatch was conducted by Jie Dai in 2001 [24]. This analysis applied a standard method of "input referral", in which the mismatch is treated as a kind of noise, which is then represented as an equivalent extra noise source at the decoder's input. If the decoder is large, then the per-bit effect of mismatch is expected to appear like Gaussian noise. Jie Dai was thus able to derive simple formulas relating the variance of mismatch to the performance of the decoder.

In 2003, the author of thesis argued that Jie Dai's method is appropriate only for the input interfaces of the decoder [91]. The interfaces are described as "feed-forward" processing stages (see Section 7.2.2 in this thesis). This result is therefore described as the *feed-forward effect*.

Jie Dai's work did not properly address the effects of mismatch in highly-interconnected networks, especially considering that the function of the network is noise reduction. An iterative decoder is appropriately described as such a highly-interconnected network, which is referred to as "lateral" processing (see Section 7.2 of this thesis).

The effect of mismatch in lateral processing was addressed by the author of this thesis in 2004 [89]. Using regular LDPC codes as a model, it was determined that a distinct *lateral effect* exists, giving rise to performance loss due to mismatch in the lateral stages. It was also determined that the lateral effect is small for low levels of mismatch. Canonical analog decoders are therefore quite robust to large amounts of mismatch.

It was also discovered that Jie Dai's feed-forward mismatch effect tends to be dominant at low levels of mismatch. Because of this result, a major conclusion of this thesis is that the input interfaces should be designed to minimize mismatch. The sum-product circuits are much more resilient, and can therefore be implemented with simpler layout techniques and smaller device sizes.

Alternative circuit topologies (2001-2004).

In 2001, Moerz and Schaefer et. al. also proposed an analog topology for implementing iterative decoding algorithms based on dual trellises [60]. This architecture would significantly reduce the complexity of analog decoders for high-rate trellis codes. The same authors also proposed a "rotating ring" architecture for analog decoders in 2003 [72]. This proposal applied analog decoding to a very new type of code, which the authors described as an "LDPC Convolutional" code. It is not known to the author whether this concept has been physically demonstrated.

In 2002, Mondragon et. al. of Texas A&M University proposed a sum-product circuit based on floating-gate MOS devices [61, 62]. These circuits were closely related to floating-gate CMOS circuits that are commonly used in neuromorphic systems [51]. Mondragon demonstrated this concept with a fabricated chip, which included a portion of an iterative decoder. Results for this chip were reported in 2003 [62].

Two unique approaches to analog decoding were suggested in 2002 and 2003 by Hemati and Banihashemi, of Carleton University [10, 11]. The first approach was designed as a solution for decoding in optical communication systems. Their second proposal was to implement the min-sum iterative decoding algorithm using a novel application of CMOS winner-take-all circuits. The min-sum approach is desirable, they argue, because the circuits operate above threshold and may therefore achieve higher speeds.

Dave Haley et. al. of the Institute for Telecommunications Research (affiliated with the University of South Australia) introduced the concept of *iterative encoding* in 2002 [39]. In 2003, he applied this idea to analog decoders [41], resulting in a mode-switching circuit capable of function both as a digital encoder and an analog decoder. This idea has undergone some refinement [40], but has not yet been physically demonstrated.

In 2003, a low-voltage sum-product topology was proposed by the author of this thesis, in collaboration with others from the University of Alberta [92]. The low-voltage topology was a modification of the canonical topology, corrected to allow proper operation with power supplies below one volt. The resulting circuits can be substituted for canonical sum-product circuits. The methodologies of Lustenberger and Jie Dai are directly applicable to the synthesis of low-voltage analog decoders using this topology.

1.2 Contributions of this thesis.

The contributions of this thesis are divided into two areas: theoretical contributions and physical demonstrations. The theoretical contributions include a few system-level results (such as the construction of efficient code graphs), analysis of the performance of large-scale analog decoders, and the design of novel circuit topologies.

Physical demonstrations include the design of the first subthreshold CMOS analog decoder (of coded length eight), serial analog interfaces for analog decoders, a length-16 analog decoder, and a length-256 analog Block Turbo decoder. All of these decoders were designed according to the basic methodology of Lustenberger.

The length-16 and length-256 decoders also made use of the "reference cell" design, which was devised by the author and described in Section 5.4 of this thesis.

1.2.1 Contributions to coding theory.

The author's analog designs have frequently implemented trellis-based decoders. The author has consequently given considerable thought to the construction of trellis graphs. In Section 3.3.5, a new method for constructing tailbiting trellises of Reed-Muller codes is introduced. This is a tailbiting adaptation of the well-known squaring construction, resulting in a lower state-complexity. It seems that tailbiting trellises are particularly well-suited for analog implementations, which could be made smaller by this result.

In Section 3.4, the author presents an algorithm for exact maximum-a-posteriori (MAP) decoding on tailbiting trellises. To the author's knowledge, this algorithm has not been explicitly reported in the literature. While the author's algorithm is in general more complex, it is shown in Section 3.4.2 that the exact tailbiting MAP algorithm is, in at least one case, less complex than standard tailbiting decoding algorithms.

In Section 4.3.3, a method is presented for applying Monte Carlo integration to the well known method of density evolution with Gaussian approximations. To the author's knowledge, this is the first application the Vegas algorithm to density evolution. The author has consequently provided a thorough examination of the precision of this algorithm, particularly when applied iteratively as examined in Section 4.3.4. This presentation provides the necessary foundation for the author's study of mismatch in analog decoders.

1.2.2 A novel "reference input" sum-product topology.

Section 5.4 presents a novel topology for sum-product circuits. This topology, called the "reference cell" for short, was devised by the author. The reference cell allows a greater diversity of sum-product circuits to be synthesized from a smaller number of standard cells. The reference cell also allows a significant reduction in transistor count for many cases. In one common case – the term-wise product of two vectors – the reference cell complexity grows linearly with vector length, whereas the canonical cell complexity grows with the square of the length.

1.2.3 A novel low-voltage sum-product topology.

Chapter 6 presents a novel topology for sum-product circuits with reduced supply voltage. The new topology, devised by the author, eliminates the need for reference voltages in analog decoding circuits. It is shown in Section 6.2.2 that the minimum supply voltage in the new topology is 0.4V lower than the minimum required voltage for canonical sum-product circuits.

Section 6.2.5 presents a circuit for iterative renormalization. It is shown that canonical renormalization is inadequate for low-voltage design. By making slight modifications, a low-voltage renormalization circuit is shown to be successful when used in a large decoding network. Iterative renormalization is a novel concept in the contexts of subthreshold analog circuits and analog decoding.

A physical demonstration of a low-voltage decoder has been designed by Nhan (Dave) Nguyen, with the author's assistance [64]. Results of this design are not presented as part of this thesis.

1.2.4 Mismatch in analog decoders.

Section 7.2 introduces a clarification of Jie Dai's mismatch analysis, dividing the decoder into feed-forward and lateral stages of operation. It is shown in that Jie Dai's results are applicable to the feed-forward stages, but not necessarily to the lateral stages. In Section 7.2.3, the method of density evolution based on the Vegas algorithm is applied to analog decoders as a means of evaluating the effect of mismatch in lateral processing stages. This is the first analytical study of mismatch in large-scale highly connected analog decoders.

The results of this examination yield new design principles for analog decoders. It is shown in Section 7.2.4 that a distinct lateral mismatch effect exists, and that it is quite small for suitably low values of mismatch. The performance of analog decoders is found to degrade dramatically when mismatch exceeds 25%. This places a very loose restriction on mismatch in lateral stages. The author therefore concludes that it is virtually unnecessary in many CMOS processes to optimize layouts for circuits in the lateral stages.

It is shown in Section 7.2.4 that the feed-forward mismatch effect is dominant for low to moderate levels of mismatch (e.g. below 20%). The author therefore concludes that standard-cell "soft-gates" are acceptable for implementation of analog decoder cores. The author further concludes that mismatch optimization is essential for feed-forward stages, including all components of the input interface. These conclusions offer the first clear design principles for managing mismatch in analog decoders.

1.2.5 Comparator offsets and yield.

Section 7.1.2 presents an analysis of how comparator offsets degrade performance. This is, to the author's knowledge, the first systematic analysis of offset effects at the output interface. It is found that the impact of offsets is quite small, as long as the offset does not exceed a fixed limit corresponding to the decoder's expected output range. The author concludes that the variation in comparator offsets is the most significant phenomenon affecting the yield of analog decoders. The author further argues that any successful built-in self-test verification for analog decoders must be able to test all comparators to confirm that their offsets are within the prescribed limit.

This result, together with the mismatch analysis, leads the author to novel conclusions about design methodology. The core methodology of Lustenberger has been well-demonstrated, in so much as it applies to lateral processing stages. Lustenberger's methods may be applied for a successful design without a great deal of new design effort. For a successful analog decoder implementation, most design effort should be focused on the interfaces. Input interfaces must be designed for minimum mismatch, and output interfaces for minimum offset. The output interface is especially critical, because it has the greatest influence on yield.

1.2.6 The first CMOS analog decoders.

The author's first physical contributions included two CMOS analog decoders, which were the first reported CMOS analog decoder designs. Both chips were fabricated in a standard 0.5μm CMOS logic process. The first of these designs was produced by the author with collaborators at the University of Utah. The sum-product circuits for this decoder were designed and simulated by the author at the schematic level. The layout of the decoder was carried out by Jie Dai. Interface circuits were designed by Woo-Jin Kim and Yong-Bin Kim. The layout of the interfaces was carried out by Woo-Jin Kim and Jie Dai. The chip was tested by the author, with the assistance of Scott Little. The project was supervised by Christian Schlegel and Chris Myers.

The first decoder chip proved difficult to test, and a flaw was found in one of the internal trellis connections [86]. Interestingly, this flaw demonstrated that analog decoders are robust to defects (even to some design defects). This also illustrated a further difficulty in testing analog decoders: though it was flawed, the chip appeared to function properly. The flaw would only have been detected by statistical tests at relatively low bit error rates. The mistake therefore called attention to an open problem of verification for analog decoders.

Chapter 8 presents design and test results for the second decoder chip, which was designed for better testability. The sum-product circuits and the decoder layout were produced by the author of this thesis. The interface was conceptualized by the author, with the assistance of Reid Harrison and Shuhuan Yu. The interface included sample-and-hold circuits, unity-gain buffers, and current comparators. All of these interface components were designed by Shuhuan Yu, who also produced the layouts. Shuhuan Yu also evaluated the interface circuits for their mismatch characteristics, and optimized the design to reduce offsets in the buffers and comparators.

Detailed simulations were carried out by Jie Dai using analog VHDL models. These simulations evaluated the decoder's performance in subthreshold, above threshold, and under the influence of device mismatch. These results were then used for comparison against test results. Supervisors in this design were Christian

Schlegel, Reid Harrison, and Chris Myers.

1.2.7 A length-16 analog MAP decoder.

Chapter 9 presents design and test results for a length-16 analog trellis-based maximum-a-posteriori (MAP) decoder. This decoder is the first design to incorporate the reference cell topology. The schematic and layout designs for this decoder were produced by the author at the University of Alberta. Interface circuits were based on the designs of Shuhuan Yu, and were redesigned by the author in order to adapt the interface architecture to a 0.18μm process.

An early schematic design of the length-16 decoder was produced by the author at the University of Utah. Some design decisions, including the use of reset circuits, were influenced by Jie Dai. Jie Dai also produced analog VHDL simulations as an initial verification of the length-16 decoder concept. The original design was intended for a 0.5μm process, to reuse interface designs from the previous chips. At the University of Utah, this design was supervised by Christian Schlegel, Chris Myers, and Reid Harrison.

The final design was completed at the University of Alberta. The final design made use of the Utah architecture, but was redesigned for implementation in a 0.18μm process. Layouts were also redrawn for the new process. It was implemented using the Tanner Tools Pro design software, and scripts were written to interface these tools with the Cadence-based design flow used by the Canadian Microelectronics Corporation. Paul Greidanus at the University of Alberta was helpful in this migration. Supervisors of this project at the University of Alberta were Christian Schlegel and Vincent Gaudet.

A test interface was also designed, in collaboration with Nhan (Dave) Nguyen at the University of Alberta. This test interface is described in Section 9.3. The test interface includes a custom circuit board and a Linux-based interface. The hardware includes a USB port, FPGA, DAC, adjustable buffers and reference sources. The software, written in C++, generates test samples, communicates them to the test board, and counts the resulting errors at the decoder chip's output. Results are displayed graphically using a custom extension of the PlPlot scientific visualization library. The hardware was designed jointly by the author and Nguyen. The FPGA program was written by Nguyen.

1.2.8 A length-256 Block Turbo decoder.

Chapter 10 presents design and test results for an analog Block Turbo decoder. This decoder was designed entirely by the author, and is perhaps the most significant contribution of this thesis. It was constructed from thirty-two instances of the length-16 analog decoder.

The Block Turbo decoder is the largest and best-performing analog decoder as of the writing of this thesis. It is the first analog decoder to demonstrate performance superior to a discrete-time iterative decoder for the same code. It is also the second analog decoder to implement a commercial standard iterative code. This decoder provides a clear proof-of-concept for large-scale analog iterative decoders.

1.3 Outline of this thesis.

The principles of error control coding and decoding algorithms are presented in Chapters 2 through 4. Analog decoding circuits are introduced in Chapter 5. The principle contributions of this thesis appear in Chapters 6 through 10. Some contributions also appear in Chapters 3, 4 and 5.

Chapter 2 presents fundamental definitions and theorems of coding theory, including the Shannon limit and the BPSK limit for Gaussian noise channels. The chapter discusses graphical representations of codes, including Tanner, constraint, and normal graphs. The sum-product decoding algorithm is given, and the extrinsic information principle is defined. Finally, the structure and design of various good codes are summarized, including Turbo codes, LDPC codes, and Block Turbo codes. These codes are compared to each other in terms of their typical performance, complexity, rates, and nearness to the Shannon limit.

In Chapter 3, trellis codes are presented as a class of graphs for linear block codes. The definition and construction of tailbiting and conventional trellis graphs are discussed. The BCJR decoding algorithm is explained as a special trellis-based form of the sum-product algorithm. The first novel results of this thesis are presented, which include a new tailbiting squaring construction for Reed-Muller and Hamming codes. A new optimal tailbiting MAP algorithm is also presented, and is shown to be less complex than BCJR for one example trellis, although it is in general more complex.

Chapter 4 presents an introduction to the structure and design of Low-Density Parity Check codes. Regular and irregular code ensembles are defined and compared against each other. The method of density evolution with Gaussian approximation is explained. A new form of the density evolution algorithm is presented, using Monte Carlo integration based on the Vegas algorithm. The precision of this new method is investigated, and the propagation of error is examined in case there are iterated applications of the method.

Chapter 5 presents an introduction to weak inversion (subthreshold) CMOS circuits. The state of the art of analog decoding is also summarized, and the "canonical" sum-product analog decoding topology is presented in detail. A novel analysis is presented for the minimum allowable supply voltage of canonical circuits. A novel "reference input" simplification is also presented, which reduces the complexity of sum-product implementations in many cases.

A novel sum-product circuit topology for low supply voltage is presented in Chapter 6. This topology is derived from the canonical sum-product circuit, which is modified to eliminate reference voltages, thereby reducing the voltage overhead. Error terms are corrected by the insertion of "dummy" devices, and attenuation is corrected by iterated renormalization, which is a novel concept. The minimum supply voltage for this topology is compared against the canonical form, and is found to give an improvement of 0.4V to 0.5V.

Chapter 7 presents a collection of analyses which pertain to the design of large-scale analog decoding architectures. The analyses are made, wherever possible, using general approaches which should apply to a variety of CMOS processes. New design principles are deduced, particularly pertaining to mismatch and comparator offsets. The Monte Carlo method of density evolution (introduced in Chapter 4) is applied to determine the effect of mismatch in large analog decoding networks.

Chapters 8, 9 and 10 present the design of and results for three analog decoders, based upon the principles elaborated in earlier chapters. Chapter 11 offers conclusions.

Chapter 2

Error-Control Codes and Decoders

2.1 Communications systems.

A simple communications system model, shown in Figure 2.1.1, consists of a transmitter, a receiver, and a channel. The transmitter's goal is to reliably communicate an information message, $\underline{u}$, to the receiver via the channel. To do this, the transmitter *encodes* $\underline{u}$ to produce a *codeword* $\underline{x}$. The purpose of this is to add some controlled redundancy to $\underline{x}$ so that errors are easily detected or corrected.

We will assume that the symbols of $\underline{u}$ are chosen from the binary alphabet $u_i \in \{0, 1\}$, and the symbols of $\underline{x}$ are chosen from the *modulated* alphabet $x_j \in \{+1, -1\}$. This notation provides an idealized model of *antipodal transmission*, in which data are transmitted by modulating the sign of some suitable pulse function $p(t)$. An example of antipodal transmission is Binary Phase-Shift Keying (BPSK) in which information is transmitted by modulating the sign of a sinusoidal function. To keep our discussion simple, we will only consider BPSK modulation.

We will consider transmission through the *Additive White Gaussian Noise* (AWGN) channel, which is a very fundamental type of channel. In the AWGN model, the transmitted codeword $\underline{x}$ is altered by adding a zero-mean, Gaussian distributed noise pattern $\underline{n}$. The receiver then observes the *channel information* $\underline{r} = \underline{x} + \underline{n}$. The receiver must then decode the channel information to produce a good *estimate* of the information message, $\widehat{\underline{u}}$.

This basic model is applicable to many types of systems for communication and storage of information. Application areas include wireless systems (from satellites

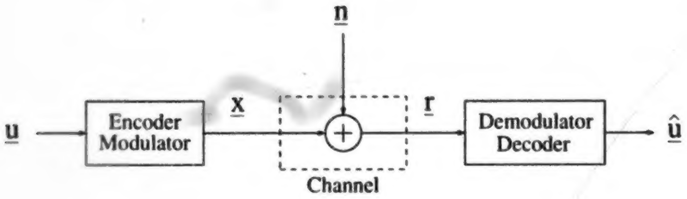


Figure 2.1.1: Communications system model.

to Bluetooth to inter-chip communication), wireline systems (telephone modems, wire networks, backplane transceivers), magnetic storage (hard disk drives and tape storage media), optical storage (CDROM media), solid-state memories (Flash RAM and DRAM) and so on. There are many other channels besides the AWGN, but equalizers are commonly used to pre-process the channel information. With this pre-processing in place, the channel behavior is made to be roughly equivalent to that of the AWGN.

2.1.1 Probabilities and log-likelihood ratios.

When the channel observations, $\underline{r}$, arrive at the receiver, they are analog in nature. The discrete nature of the original message $\underline{x}$ is corrupted by the analog channel noise. The resulting samples may be represented in several ways.

In a hard-decision receiver, the samples in $\underline{r}$ are immediately resolved to a sequence of bits, $\hat{\underline{x}}$. This can be described as analog-to-digital conversion with one bit resolution. The decoder must then *detect* errors in $\hat{\underline{x}}$ and attempt to *correct* them by guessing which bits are most likely to be mistakes.

In a soft-information receiver, the analog information in $\underline{r}$ is preserved, often by digitizing $\underline{r}$ with a resolution of several bits. To be useful for decoding, this analog information must be translated into *probability* information. The decoder may then perform a detailed probability calculation to determine which bit-decisions should be made.

One way to represent a sample r_i is through a probability mass $\underline{\rho}_i$. For antipodal transmission on the AWGN channel, $\underline{\rho}_i$ is defined as

$$\underline{\rho}_i = \left\langle \begin{array}{c} P(\mathbf{x}_i = -1 \mid r_i) \\ P(\mathbf{x}_i = +1 \mid r_i) \end{array} \right\rangle. \tag{2.1.1}$$

Each $\mathbf{r}_i$ is Gaussian distributed with mean ± 1, and with variance $N_0/2$, where N_0 is the channel's noise power. We write the pdf of $\mathbf{r}_i$ as $g_\mu(r)$, where $\mu = \pm 1$ refers to the mean. Let the vector $\underline{\rho}_i$ be indexed by the mean, μ. Then $\rho_i(\mu) \propto g_\mu(r_i)$. Because $\underline{\rho}_i$ is a probability mass, we know that $\rho_i(+1) + \rho_i(-1) = 1$. To compute $\underline{\rho}_i$, we therefore calculate the Gaussian probability densities given $\mu = +1$, and $\mu = -1$, respectively, and then normalize the resulting vector.

The probability information related to $\underline{r}$ can also be expressed using log-likelihood ratios (LLRs). LLRs provide a dual domain to that of probabilities, where computation is often simpler. A log-likelihood ratio X_i for the binary random variable $\mathbf{x}_i$ is defined as

$$X_i = \ln\left(\frac{P(\mathbf{x}_i = -1)}{P(\mathbf{x}_i = +1)}\right). \tag{2.1.2}$$

LLRs are particularly convenient for antipodal transmission on AWGN channels. It can be easily verified from (2.1.2) that the LLR for a transmitted symbol $\mathbf{x}_i$ given a received sample r_i is equal to

$$X_i(r_i) = \frac{4}{N_0} r_i. \tag{2.1.3}$$

The probability mass $\underline{\rho}_i$ can be directly obtained from X_i through the transformation

$$\begin{aligned} P(\mathbf{x}_i = -1) &= \frac{e^{X_i}}{1 + e^{X_i}} && (2.1.4) \\ P(\mathbf{x}_i = +1) &= \frac{e^{-X_i}}{1 + e^{-X_i}}. && (2.1.5) \end{aligned}$$

When the decoder is provided information in one of these formats, it is possible to produce an optimal estimate of the original data, $\hat{\underline{u}}$. Both probability and LLR representations allow various sub-optimal approximations which provide good performance while simplifying the computation. The choice of format depends on the needs of the particular algorithm and architecture.

Table 2.1: List of notation for communications and coding.

E_b	Energy per information bit. This may be in Joules/bit, but we need assume no particular energy units.
E_s	Energy per transmitted (modulated) symbol. We will assume that $E_s = 1$ (normalized signal energy).
N_0	Power spectral density of the channel noise. The channel noise is Gaussian distributed with variance σ^2. $N_0 = 2 \cdot \sigma^2$.
SNR	Signal-to-noise ratio in dB, always defined as $10 \cdot \log_{10}\left(\frac{E_b}{N_0}\right)$.
S	Signal-to-noise energy ratio (unitless), defined as $\frac{E_b}{N_0}$.
W	The bandwidth allowed for the transmission. We will typically assume $W = 1$ (normalized bandwidth).
R	The transmission rate in units of information bits per symbol. Thus $E_s = R \cdot E_b$. $(R < 1)$.
C	The channel's capacity: $R < C$ for reliable communication [78].

2.2 The Shannon limit.

Error control coding can be used to reduce the probability of error in a communication system such as Figure 2.1.1. In 1948, Claude Shannon demonstrated that error-free communication is achievable through the use of error-control codes, as long as the transmission rate does not exceed the channel's *Capacity* [78]. The transmission rate refers to the number of information bits per channel use. The rate is always less than 1 for non-trivial codes, because every bit's information is "spread out" across more than one coded bit.

Before explaining the Shannon limit, some additional definitions are in order. Table 2.1 provides a list of symbols and associated definitions for the fundamental quantities of Shannon's communication theory. With these definitions in hand, we may describe the capacity of the AWGN channel. If the AWGN channel has continuous inputs and continuous outputs, then the capacity limit is

$$R < W \cdot \log_{10}\left(1 + \frac{R}{W}S\right). \tag{2.2.1}$$

Under the best possible circumstances, we have no bandwidth limitation, so $W \to \infty$. The bound (2.2.1) is more useful when expressed as a limit on S. To do this, we define $r = \frac{R}{W}$ and solve for S. Taking the limit, we arrive at

$$\begin{aligned} S &> \lim_{r\to 0} \frac{2^r - 1}{r} && (2.2.2)\\ \Rightarrow S &> \ln(2) && (2.2.3)\\ \Rightarrow SNR &> -1.6\text{dB}. && (2.2.4) \end{aligned}$$

This result dictates that, no matter what coding or modulation we use, and no matter how much bandwidth is available, we can never transmit reliably with $SNR < -1.6$dB.

In a band-limited system, for which we assume $W = 1$, this limit becomes a function of the rate:

$$SNR > 10 \cdot \log_{10}(2^R - 1) - \log_{10}(R) \tag{2.2.5}$$

For antipodal modulations such as BPSK, the Shannon limit becomes even more restrictive. The complete solution is somewhat ugly:

$$\begin{aligned} C &= -\int_{-\infty}^{\infty} \phi_{N_0}(y) \log_2[\phi_{N_0}(y)]\,dy - \frac{1}{2}\log_2[e\pi N_0] && (2.2.6)\\ \text{where } \phi_{N_0}(y) &= \frac{1}{2}\left\{g\left(\frac{y-1}{\sqrt{2N_0}}\right) + g\left(\frac{y+1}{\sqrt{2N_0}}\right)\right\} && (2.2.7) \end{aligned}$$

and where $g(x)$ is the Gaussian probability density function with mean zero and unit variance. The integral (2.2.6) can be carried out numerically, and provides C as a function of $\frac{E_s}{N_0}$. To express this result as a bound on SNR, we solve as follows:

$$\begin{aligned} R &< C\left(\frac{E_s}{N_0}\right)\\ \Rightarrow C^{-1}(R) &< \frac{R \cdot E_b}{N_0}\\ \Rightarrow SNR &> 10\log_{10}\left[C^{-1}(R)\right] - 10\log_{10}[R]. && (2.2.8) \end{aligned}$$

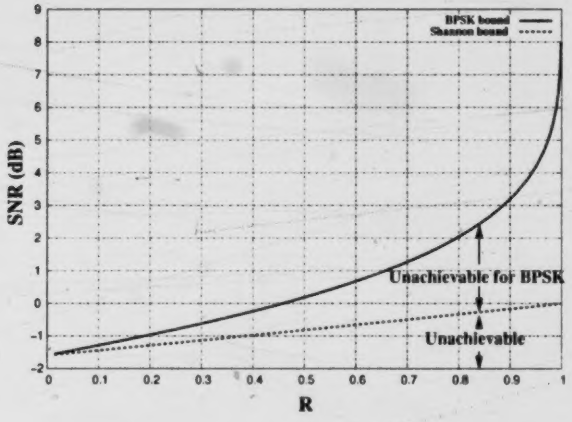


Figure 2.2.1: Shannon and BPSK limits.

We refer to (2.2.8) as the *BPSK limit*, which is more restrictive than the Shannon limit described by (2.2.5). The BPSK and Shannon limits are plotted in Figure 2.2.1. The exact BPSK limit is a somewhat complicated calculation. For the purposes of a thumbnail estimate, the following curve is a near fit to the BPSK limit:

$$\mathcal{B}(R) \approx -2 + 3R + \frac{0.4}{1.07 - R} \text{ (dB)}. \tag{2.2.9}$$

BPSK uses only two distinct signals in its modulation alphabet. There are other modulation schemes which use substantially larger alphabets. A modulation with a larger alphabet typically has its limit closer to the Shannon limit than BPSK, but may add substantial complexity to the communication system. We will always assume BPSK modulation in our discussion on error control codes. The reader should bear in mind, though, that a code which performs near the BPSK limit may still perform far from the actual Shannon limit if it has a high rate.

2.3 Linear binary block codes.

A *linear block code* $\mathcal{C}$ is a set of vectors called *codewords* which are of fixed length n, and which constitute a linear space. A linear code is defined over some finite Galois field. For our purposes it will suffice to consider only the binary field $\mathbb{Z}_2$. For any two codewords $\underline{x}_1, \underline{x}_2 \in \mathcal{C}$, the vector sum $\underline{x}_1 + \underline{x}_2$ is also in $\mathcal{C}$. Also, $\mathcal{C}$ contains an identity codeword (the all-zero codeword), written $\underline{x}_0$.

The linear code space $\mathcal{C}$ is represented by the $k \times n$ *generator matrix* G. The rows of G are codewords which form a basis for $\mathcal{C}$. If $\underline{u}$ is a k-dimensional binary row vector, then a unique n-dimensional codeword $\underline{x}$ is obtained from $\underline{u}$ by the matrix product

$$\underline{x} = \underline{u} \cdot G. \tag{2.3.1}$$

G forms a one-to-one mapping from binary vectors of length k to binary vectors of length n. Thus $G : \mathbb{Z}_2^k \rightharpoonup \mathbb{Z}_2^n$ and $G : \mathbb{Z}_2^k \mapsto \mathcal{C}$, where the symbols '$\rightharpoonup$' and '$\mapsto$' denote mappings *into* and *onto*, respectively. In more plain English, the entire space $\mathbb{Z}_2^k$ is mapped to $\mathcal{C}$, and $\mathcal{C}$ is a subset of $\mathbb{Z}_2^n$.

A linear code derives its strength from the fact that the set of codewords $\mathcal{C}$ is embedded in a much larger linear space. If $\underline{x}$ is a codeword, and all of the nearest neighbors of $\underline{x}$ in $\mathbb{Z}_2^n$ are not codewords, then an error event is less likely to make $\underline{x}$ look like a different codeword. A simple but useful measure of distance is the *Hamming distance*, $h(\underline{x}_1, \underline{x}_2)$, defined as the number of positions in which $\underline{x}_1$ and $\underline{x}_2$ differ, where $\underline{x}_1, \underline{x}_2 \in \mathbb{Z}_2^n$.

Another useful measure is the *Hamming weight* of a codeword $\underline{x}$, defined as $h(\underline{x}_0, \underline{x})$. An important parameter in determining the error-correcting power of a code is its minimum distance, $d_{\min}$, defined as the minimum of $h(\underline{x}_1, \underline{x}_2)$ between any two codewords $\underline{x}_1, \underline{x}_2 \in \mathcal{C}$. For linear codes, $d_{\min}$ is equal to the smallest non-zero Hamming weight among the codewords of $\mathcal{C}$.

The code described above is usually referred to as an $(n, k, d_{\min})$ code, or often just an (n, k) code. For a linear block code, it is easy to tell whether a given sequence is a codeword or not.

The *parity-check matrix* H for $\mathcal{C}$ is defined as the generating matrix for the *null space* of $\mathcal{C}$. That is, H is the matrix for which $GH^{\mathrm{T}} = 0$. The rows of H are a basis for a linear space in which every vector is orthogonal to any codeword. Therefore, if $\underline{x}$ is a codeword,

$$\underline{x} \cdot H^{\mathrm{T}} = 0. \tag{2.3.2}$$

If, for some $\underline{x} \in \mathbb{Z}_2^n$, $\underline{x} \cdot H^{\mathrm{T}} \neq 0$, then $\underline{x} \notin \mathcal{C}$. The parity-check matrix thus provides a convenient way to test whether a given sequence is a codeword. If $\underline{x} \cdot H^{\mathrm{T}} = 0$ then we say that all parity checks are *satisfied*.

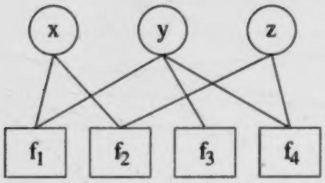


Figure 2.3.1: Example factor graph for $f(x, y, z) = f_1(x, y) \cdot f_2(x, z) \cdot f_3(y) \cdot f_4(y, z)$.

2.3.1 Factor graphs and Normal graphs.

There are various ways to graphically represent the linear space of codewords, including *constraint graphs* and *trellis graphs*. These are subclasses of the more general *factor graphs* [48]. We will consider a factor graph as a representation of *Boolean constraint functions* on *discrete domains*. Such a function takes arguments from a set of discrete variables, and returns a Boolean result, indicating whether those variables *satisfy* some formal constraint.

A factor graph consists of *nodes* and *edges*. There are two types of nodes: *variable nodes* and *function* (or *constraint*) *nodes*. The overall graph represents a function $f(X)$, where X is the set of all variables on which f depends. It is assumed that $f(X)$ can be factored into a sequence of functions whose domains are subsets of X, i.e.

$$f(X) = f_1(X_1) \cdot f_2(X_2) \cdots f_m(X_m) \tag{2.3.3}$$

where each $X_j \subset X$.

For each $f_j(X_j)$, there is a corresponding function node in the graph. For each distinct variable in X, there is a corresponding variable node. An edge exists between a function node $f_j(X_j)$ and a variable node x_i if and only if $x_i \in X_j$. Edges exist only between variable and function nodes. An example factor graph is shown in Figure 2.3.1. It is conventional to represent variable nodes with circles and function nodes with squares.

It is possible to construct a factor graph with *hidden variables* through variable substitutions. A variable h in a factor graph of $f(X)$ is called "hidden" if $h \notin X$. The hidden variable is a function of variables in X, e.g. $h = \rho\left(X_j\right)$ for some hidden function ρ and for some $X_j \subset X$. As an example, the function $f(x, y, p, q) =$

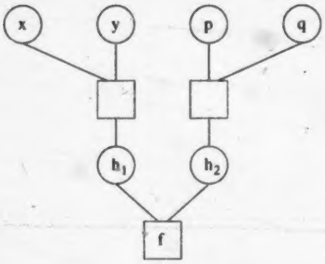


Figure 2.3.2: Factor graph with hidden variables.

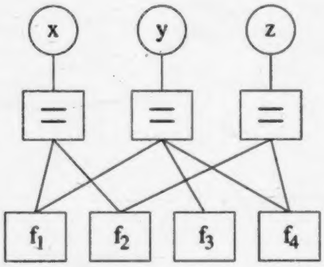


Figure 2.3.3: Normal graph corresponding to Figure 2.3.1.

$(x+y)\cdot(p+q)$ can be rewritten as $f(h_1, h_2)$, where $h_1 = x+y$ and $h_2 = p+q$. The corresponding factor graph is shown in Figure 2.3.2.

A slightly different approach to factor graphs is provided by Forney's *normal graphs* [31]. A normal graph allows direct connections between function nodes, but requires that every variable node connect to only a single function node. The purpose of this distinction is to provide an explicit representation of all functional relationships, while omitting possible hidden variables from the graph. Examples of normal graphs are shown in Figures 2.3.3 and 2.3.4.

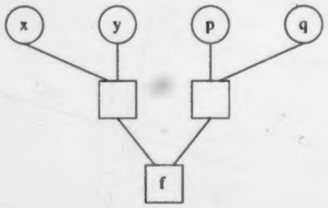


Figure 2.3.4: Normal graph corresponding to Figure 2.3.2.

An important quality of a factor graph is the presence or absence of *cycles*. A *path* from node x to node y is a connected sequence of edges and nodes which begins at x and ends at y, in which no edge is traversed more than once. If $\mathcal{G}$ is a factor graph, and x is a node in $\mathcal{G}$, then a path from x to x is said to be a *cycle*. $\mathcal{G}$ is said to be a *loopy graph* if it contains more than one cycle. If $\mathcal{G}$ contains a single cycle, then it is called a *tailbiting graph*. If $\mathcal{G}$ contains no cycles, then $\mathcal{G}$ is said to be a *tree*.

2.3.2 Tanner graphs.

An important representation of a binary block code is the factor graph of its parity-check equation, $\underline{x} \cdot H^{\mathrm{T}} = 0$. Let r be the number of rows in H, and let $\underline{h}_j$ be the j^{th} row of H. The parity-check equation consists of r functions of the form $f_j\left(X_j\right) = \underline{x} \cdot \underline{h}_j$, where $X_j = \left\{x_i \,|\, h_{ji} = 1\right\}$. Each function $f_j\left(X_j\right)$ is the $\mathbb{Z}_2$ sum over all variables in X_j. The resulting factor graph consists of n variable nodes and r function nodes. Each of the function nodes, called *parity-check* nodes, represents $\mathbb{Z}_2$ addition. Such a graph is known as a *Tanner graph* or *constraint graph* for H, written $\mathcal{T}(H)$ [84]. When constructed in this manner, there is a one-to-one correspondence between the rows of H and the single-parity-check subgraphs of $\mathcal{T}(H)$.

A constraint graph is a more restricted kind of graph than a factor graph. The factor graph represents the factored structure of a function $f(X)$. The constraint graph adds one further condition: an implicit constraint $f(X) = 0$ (or sometimes $f(X) =$ 'true', or whatever Boolean constraint is most convenient). Thus if one of the factors is $f_j(x, y, z)$, such a constraint allows us to determine the value of x based on y and z. We shall also see that the probability mass of x can be determined based on the masses of y and z. This calculation, called probability propagation, is the basis of powerful iterative decoding algorithms.

As an example, consider the parity-check matrix

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

for which the corresponding Tanner graph is shown in Figure 2.3.5.

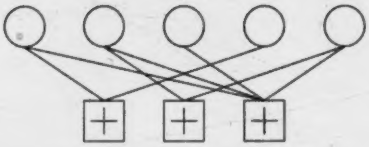


Figure 2.3.5: Example Tanner graph.

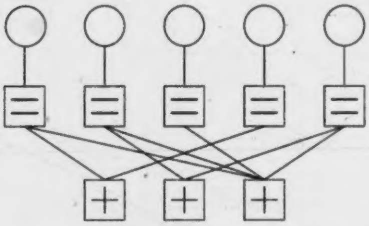


Figure 2.3.6: Normalized Tanner graph corresponding to Figure 2.3.5.

Note that, for any linear combination of parity checks $\underline{h}' = \underline{h}_j + \underline{h}_k$ (recall that addition is in $\mathbb{Z}_2$), $\underline{h}'$ is also a parity check for code $\mathcal{C}$. Let $\mathcal{T}'(H)$ be the graph which results from appending $\underline{h}'$ to $\mathcal{T}(H)$. It is sometimes useful to add redundant nodes in this way as a means of modifying the graph's structure. $\mathcal{T}'(H)$ is still a valid representation of the parity checks of $\mathcal{C}$, although it no longer has a one-to-one correspondence to H.

To be appropriately general, then, we define a Tanner Graph as any bipartite graph, consisting of variable nodes and parity-check nodes, in which each edge connects one variable node to one parity check node. We may then speak of $\mathcal{T}(H)$ (the Tanner graph induced by H) and $H'(\mathcal{T})$ (the parity-check matrix H' induced by $\mathcal{T}$). $\mathcal{T}$ is a representation of $\mathcal{C}$ if all the rows of H are obtainable through linear combinations of the rows of $H'(\mathcal{T})$.

The normalized form of the Tanner graph, $\mathcal{N}$, is easily obtained from $\mathcal{T}$ by substituting *equality nodes* for all variable nodes. The equality node denotes a constraint of equality among all connected edges. Variable nodes are reinserted above the equality nodes, which are connected by a single edge. An example normalized Tanner graph is shown in Figure 2.3.6.

2.4 Decoding algorithms.

In the communications model of Figure 2.1.1, the receiving device must somehow infer the intended message $\underline{u}$ based on channel observations $\underline{r}$. The strength of an error control system is only partly determined by the geometric characteristics of the code (e.g. $d_{\min}$). Performance is also limited by the quality of the decoding algorithm.

Description of decoding algorithms requires discussion of probabilities, for which we require some refined notation. A bold-face letter $\underline{\mathbf{u}}$ shall indicate a random variable, and an ordinary $\underline{u}$ shall indicate a particular (deterministic) value. We will use the notation $\widehat{\underline{u}}$ to indicate an *estimate* of $\underline{\mathbf{u}}$.

2.4.1 MAP versus ML decoding.

A *maximum a posteriori* (MAP) decoder determines the estimate $\widehat{\underline{u}}$ which maximizes

$$P(\underline{\mathbf{u}} = \widehat{\underline{u}} \mid \underline{r}). \tag{2.4.1}$$

A *maximum likelihood* (ML) decoder determines the estimate $\widehat{\underline{u}}$ which maximizes

$$P(\underline{\mathbf{r}} = \underline{r} \mid \widehat{\underline{u}}). \tag{2.4.2}$$

These rules are equivalent if and only if the information messages $\underline{u}$ are all equally probable. If this is not the case, then the decoder must know the probability mass of $\underline{\mathbf{u}}$ in order to calculate the MAP solution.

A MAP decoder can be "easily" implemented by computing $P(\underline{\mathbf{u}} = \widehat{\underline{u}} \mid \underline{r})$ for every possible message $\hat{\underline{u}}$. This procedure is referred to as *exhaustive search* decoding. Exhaustive searches are absurdly complex and impossible to implement efficiently for all but the most trivial error control codes.

In practice, approaches to decoding algorithms are informed by the MAP and ML rules. The strongest decoders, however, are not exact ML or MAP decoders. Their strength arises from a compromise between exactness and efficiency. The great achievement of iterative decoders is not just their high performance, but their manageable complexity.

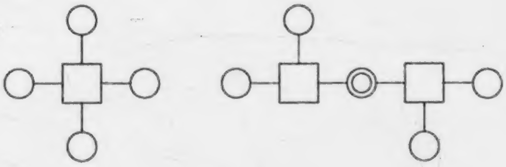


Figure 2.4.1: Hidden variable insertion.

Most iterative decoding algorithms are instances of *probability propagation* on factor graphs. Probability propagation is carried out by the sum-product algorithm. If the factor graph is a tree, then this algorithm is equivalent to MAP decoding. Often, though, loopy graphs provide much more efficient implementations. The algorithm is no longer exact on loopy graphs, but in many cases it can be fine-tuned to produce near-MAP results.

2.4.2 The sum-product algorithm.

The *sum-product algorithm* (or *SP algorithm*) is a general framework for implementing probability propagation on graphs [48]. The purpose of the algorithm is to compute global conditional probabilities using only local constraints. Constraints are expressed by factor graphs, which were examined in Section 2.3.1. The sum-product algorithm is implemented by *message passing* between simple processing nodes. Each node updates its outgoing messages based on messages received from adjacent nodes. Messages between nodes consist of probability information.

For most factor graphs, we only need to describe local processing in nodes with three edges. Constraints on more than three variables can usually be reduced by insertion of a hidden variable, as in Figure 2.4.1. The graph for a three-variable constraint is shown in Figure 2.4.2. The function f expresses a relationship between random variables $\mathbf{x}$, $\mathbf{y}$ and $\mathbf{z}$ which can take values from discrete alphabets $\mathcal{A}_x$, $\mathcal{A}_y$, $\mathcal{A}_z$, respectively. We say that $f(x, y, z)$ is a *Boolean constraint* on (x, y, z) if $f(x, y, z) \in \{0, 1\}$ for all x, y, and z. We say that the constraint f is *satisfied* if and only if $f(x, y, z) = 0$.

The local operations of the sum-product algorithm are described as follows. The constraint f is mapped to a *processing node* which receives probability masses for

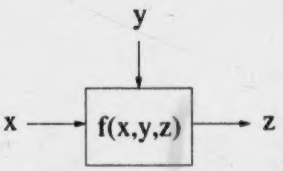


Figure 2.4.2: Function node for a Boolean constraint on three variables.

variables **x** and **y**. These variables are assumed to be independent of each other. The processing node then computes the probability mass of **z** based on the constraint f and the masses of **x** and **y**. Let $\mathcal{S}_f$ be the set of combinations of (x, y, z) for which $f(x, y, z)$ is satisfied.

Let $\mathcal{S}_f(j)$ be the subset of $\mathcal{S}_f$ for which $z = j$, where $j \in \mathcal{A}_z$. We then compute, for each j, the function

$$P(\mathbf{z} = j) = \eta \cdot \sum_{(k,l) \in \mathcal{S}_f(j)} P(\mathbf{x} = k) \cdot P(\mathbf{y} = l) \tag{2.4.3}$$

where $k \in \mathcal{A}_x$ and $l \in \mathcal{A}_y$, and η is any non-zero constant real number. The constant η is typically chosen so that $\sum_j P(\mathbf{z} = j) = 1$. In principle, though, η has no effect on the accuracy of the algorithm.

If the probability masses of variables **x** and **y** are conditional, then their conditionality is inherited by **z** after application of (2.4.3). Thus the decoder begins with a set of symbol probabilities $P(\mathbf{x_i} = j \,|\, r_i)$, conditioned on the channel observations r_i. Through repeated iterations of (2.4.3), the decoder ultimately arrives at an estimate of $P(\mathbf{u_i} = j \,|\, \underline{r})$ for every information bit $\mathbf{u_i}$.

The local computation (2.4.3) is the heart of the sum-product algorithm. A complete sum-product decoder consists of many interconnected instances of (2.4.3). The propagation rule (2.4.3) must be implemented separately for each edge of a node. Thus each three-edged node in a factor graph requires three instances of (2.4.3), as illustrated in Figure 2.4.3.

The need for independent sum-product implementations in Figure 2.4.3 is a consequence of the *extrinsic information principle*. The extrinsic information principle describes a necessary condition on message processing in any decoding network. It

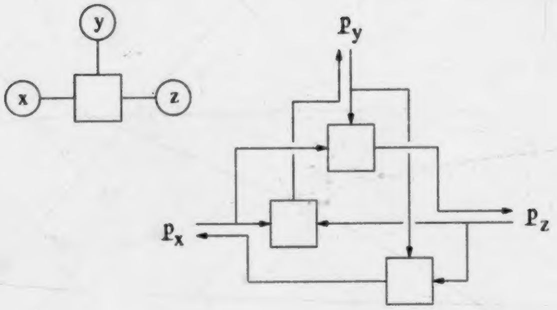


Figure 2.4.3: Implementation of probability propagation in a node of degree three.

is essential to the successful implementation of a complete sum-product decoder.

Definition 2.4.1. Extrinsic Information Principle. The output message from node *N* on edge *E* must be independent of any message received by node *N* on edge *E*.

□

If this principle is upheld globally, then a sum-product decoder is equivalent to a MAP decoder. Note that if there are cycles in the factor graph, then the extrinsic information principle is violated (in a global sense). If the cycle is long (i.e. if it consists of sufficiently many nodes) then the principle is only "weakly" violated, in that the edge's output is only weakly correlated with its input.

Since the introduction of Turbo Codes, a variety of methods have been suggested for meeting the extrinsic information principle. In the original Turbo Codes, extrinsic information is obtained by dividing the input messages from the output messages. This is a somewhat circuitous approach. If normal graphs are used, the equality node always intervenes to ensure that the extrinsic information principle is met. No rule other than (2.4.3) need ever be applied.

2.5 Good codes.

We are now able to discuss the structure of some of the best classes of error control codes for the AWGN channel. We loosely define a "good code" as one whose performance approaches the Shannon and/or BPSK limits "relatively quickly" as the block length is increased. We may sometimes also say that a code is "good" if it has an efficient decoder which performs significantly better than other codes of comparable complexity. Powerful codes are best described graphically, and we shall see that their strength is owed to graph-based decoding algorithms.

Important classes of "good codes" include Turbo codes, Low-Density Parity Check (LDPC) codes, and Block Product (aka Turbo Product) codes. All of these codes are constructed by "stitching together" simpler component codes in some way. We will briefly describe the structures of some good codes, and describe the ways in which they implement the sum-product algorithm for decoding.

Alternative (non-sum-product) algorithms are known for most of these codes. These alternatives are either approximations to the sum-product algorithm, or are based on other sub-optimal approaches. Such algorithms may provide simplifications in particular decoder implementations (and may in some cases be more widely used than the proper sum-product algorithm because of improved efficiency or designability). We will only discuss sum-product implementations, because other algorithms do not currently shed any light on analog implementations.

2.5.1 Parallel Concatenated Convolutional Codes: Turbo Codes.

Parallel Concatenated Convolutional Codes (PCCC) are the original Turbo Codes [16]. These codes are constructed from a pair of trellis codes, $\mathcal{C}_1$ and $\mathcal{C}_2$, which are called *component codes*. Component encoders for $\mathcal{C}_1$ and $\mathcal{C}_2$ are written E_1 and E_2, respectively. Component decoders are D_1 and D_2.

The component codes are used to separately encode the same information sequence $\underline{u}$, except that the order of the bits in $\underline{u}$ is *scrambled* before being encoded by E_2. The bits are scrambled by a pseudo-random *interleaver* (or permuter), which is typically denoted by the symbol Π. The two encoders produce distinct sequences of parity-check bits $\underline{p}_1$ and $\underline{p}_2$. The information sequence is multiplexed together with all or some of the two parity sequences to form the transmitted data stream $\underline{x}$. If only some of the bits from $\underline{p}_1$ and $\underline{p}_2$ are used (for example, half of the bits are selected from $\underline{p}_1$ and the other half from $\underline{p}_2$), then we say the codeword $\underline{x}$ is *punctured*. This encoding procedure is illustrated in Figure 2.5.1.

The decoder for a PCCC Turbo Code consists of two component trellis decoders

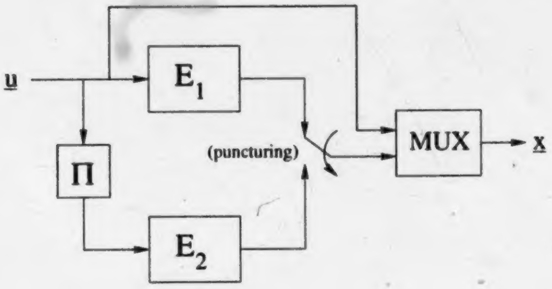

u
E1
Π
(puncturing)
E2
MUX
x

Figure 2.5.1: PCCC Turbo Code encoder.

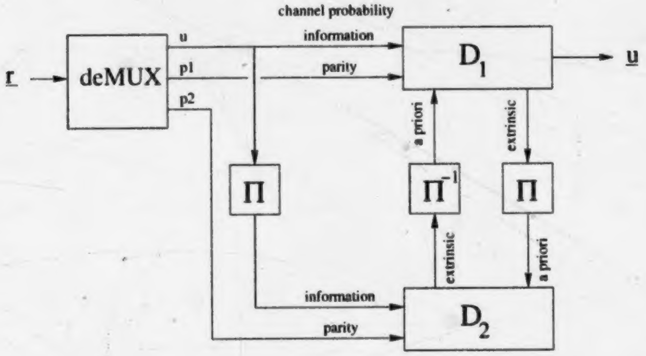

r
deMUX
u
p1
p2
channel probability
information
parity
D1
u
a priori
extrinsic
Π
Π⁻¹
Π
extrinsic
a priori
information
parity
D2

Figure 2.5.2: Decoder for a PCCC Turbo Code.

which implement soft-output algorithms. Soft-output algorithms include the BCJR algorithm [9], the SOVA algorithm [37], and a variety of suboptimal variations [73]. The important feature of a Turbo Decoder is the the component decoders *collaborate* with each other. This setup is illustrated in Figure 2.5.2. D_1 and D_2 each receive their respective data streams for decoding. Any punctured bit is replaced by a probability of 0.5. D_1 and D_2 then decode assuming uniform *a priori* probabilities for the information bits. After the symbol probabilities are all calculated, D_1 and D_2 *decode the data again*, only this time D_1 uses the results from D_2 as an estimate of the *a priori* bit probabilities, and D_2 similarly uses the results from D_1. This process is *iterated* many times until reliable bit decisions are obtained.

This decoding scheme works because $\underline{p}_1$ and $\underline{p}_2$ are approximately independent of each other, thanks to the interleaver. The extrinsic information from D_1 is decorrelated from the channel information available to D_2, and vice versa. The two

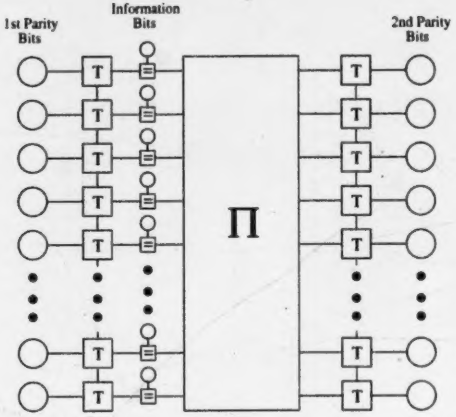


Figure 2.5.3: Factor graph for a PCCC Turbo Code.

decoders are therefore able to exchange information without destructive feedback. This observation led to the generalized extrinsic information principle, which is fundamental to the success of iterative decoders.

It is also possible to represent a PCCC decoder as a factor graph, as shown in Figure 2.5.3. Both component codes in this graph are rate-$\frac{1}{2}$. The component trellis decoders are depicted vertically on the left and right of the graph, separated by the large interleaver, labeled Π. Individual trellis sections are indicated by the boxes labeled 'T'. Notice that the parity bits are located at "leaf" positions in the graph, while the information bits are nested in between the component decoders. The iterative Turbo decoding algorithm is equivalent to the sum-product algorithm on the factor graph of Figure 2.5.3.

The performance of the original PCCC Turbo Code is shown in Figure 2.5.4. Note that the BER reaches an abrupt floor at around 10^{-6}. This is a common characteristic of PCCC Turbo decoders. Numerous techniques have been explored to lower the error floor of Turbo codes. Current Turbo codes have much lower floors, but it remains a significant challenge in Turbo code design.

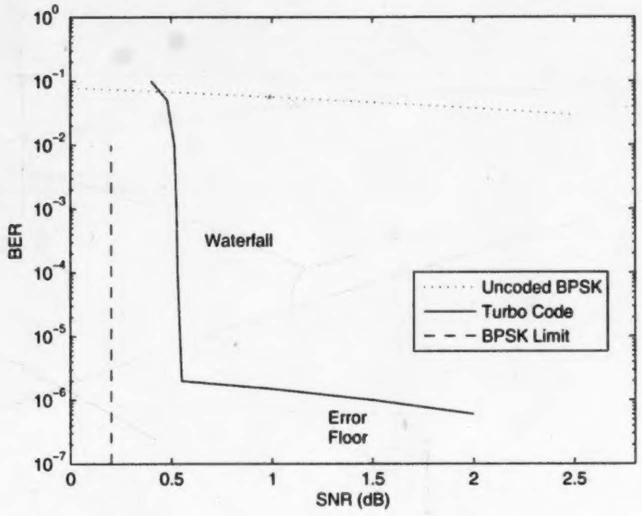


Figure 2.5.4: Performance of the original rate-1/2 PCCC Turbo Code, with length 65536 and 18 decoding iterations [15]. The BPSK limit is also indicated.

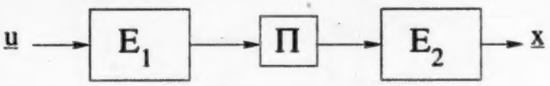


Figure 2.5.5: Encoder for an SCCC Turbo Code.

2.5.2 Serially Concatenated Convolutional Codes.

Shortly after the unveiling of PCCC Turbo Codes, a second class of Turbo Codes was introduced: Serially Concatenated Convolutional Code (SCCC) Turbo Codes [12, 81]. The difference between SCCC and PCCC Turbo Codes is simple: instead of encoding the same data independently, E_2 encodes the interleaved output from E_1. An SCCC encoder is illustrated in Figure 2.5.5.

Decoding of SCCC Turbo Codes is similar to that of PCCC codes. A block diagram for the decoder is shown in Figure 2.5.6. SCCC Turbo Codes are in some ways simpler and more elegant than PCCC codes. There is still some argument as to which class offers superior performance (or whether either class can be said to be generally better than the other).

Factor graphs for systematic and non-systematic SCCC Turbo Codes are shown

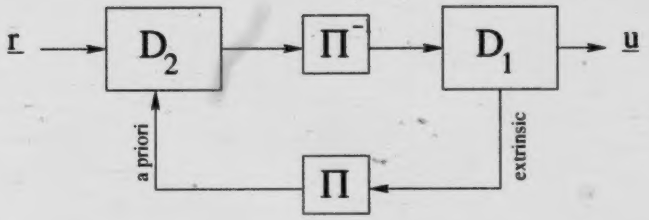
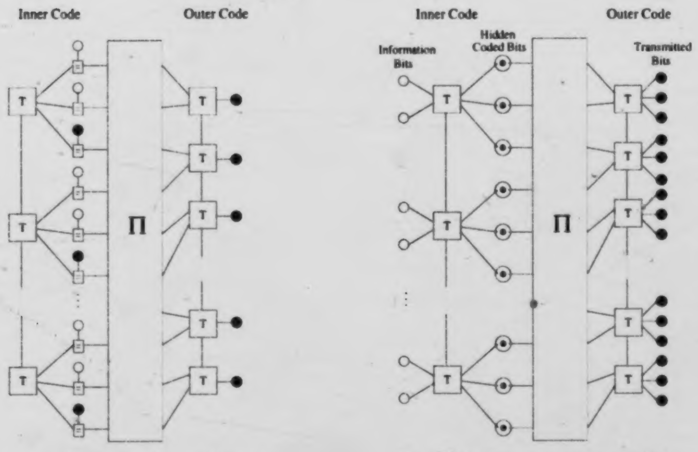

r
D2
Π⁻
D1
u
a priori
extrinsic
Π

Figure 2.5.6: Decoder structure for SCCC Turbo Codes.


Inner Code
Outer Code
T
Π
Inner Code
Hidden Coded Bits
Outer Code
Information Bits
Transmitted Bits
T
Π

a. Systematic code.

b. Non-systematic code.

Figure 2.5.7: Factor graphs for SCCC Turbo Codes.

in Figure 2.5.7. In the systematic graph, filled circles represent parity bits, and open circles represent information bits. In the systematic code, all variables are transmitted. In the non-systematic code, the open circles represent information bits. The double-circles are the intermediate coded bits at the output of E_1. These are called "hidden" bits because they are never observed at the input or output of the encoder or decoder. They are purely internal. The filled circles are the final coded bits which are transmitted. All component codes in Figure 2.5.7 are rate-$\frac{1}{3}$.

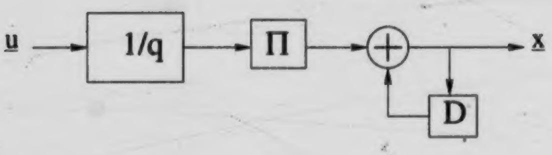


Figure 2.5.8: Repeat-Accumulate encoder.

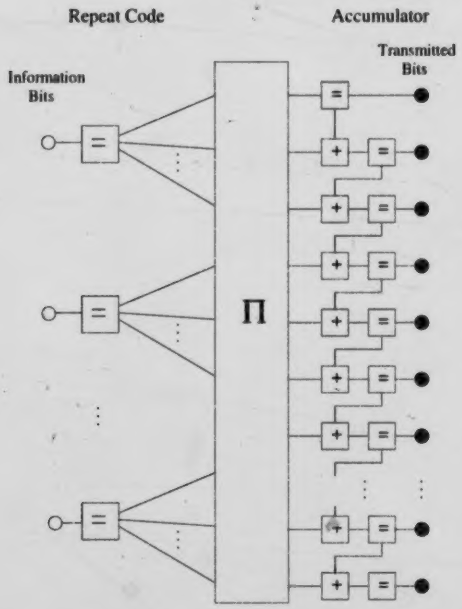


Figure 2.5.9: Repeat-Accumulate code factor graph.

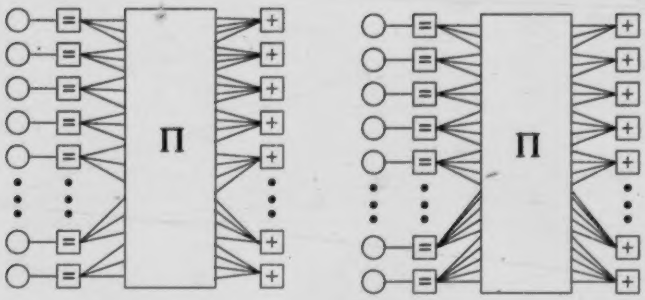
2.5.3 Repeat-Accumulate codes.

One simple variety of SCCC Turbo Codes are the Repeat-Accumulate (RA) codes. These codes are extremely simple. The inner code is a *repetition code*[1] of rate $\frac{1}{q}$ and the outer code is a rate-one accumulator. An RA encoder is illustrated in Figure 2.5.8. The corresponding factor graph is shown in Figure 2.5.9.

Repetition codes and accumulators are fairly trivial, weak codes by themselves. Yet some RA codes have performance within 1dB of the Shannon Limit. This is a dramatic demonstration of the power of iterative decoding: two extremely weak codes can be combined to produce a very good code, with very low complexity.

---

[1] A rate-$\frac{1}{q}$ repetition code takes a single bit as input and produces q copies of that bit as output.



Regular Code Irregular Code

Figure 2.5.10: LDPC code factor graphs.

2.5.4 Low-Density Parity Check codes.

Low-Density Parity Check (LDPC) codes are large binary block codes with sparse parity-check matrices. They are divided into the classes of ***regular*** and ***irregular*** LDPC codes. A (d_r, d_c) regular LDPC code has a parity check matrix H in which each row has d_r ones and each column has d_c ones. The number of ones in a column/row of H is said to be the *degree* of that column/row. Thus we say that d_r is the *row degree* of H and d_c is the *column degree* of H. The pattern of non-zero entries in H is more-or-less random. The total density of ones in H is small, hence the name "low-density" parity-check code.

An irregular LDPC code is one in which the row and column degrees are not fixed to a single value. The *degree distribution* of columns and rows in H is specified by a pair of probability masses $\underline{\tilde{\lambda}}$ and $\underline{\tilde{\rho}}$, respectively. Each $\tilde{\lambda}_i$ is in the range [0, 1], denoting the fraction of columns with degree i. Similarly, each $\underline{\tilde{\rho}}_j$ denotes the number of rows with degree j.

As shown in Section 2.3.2, the parity-check matrix H corresponds to a bipartite Tanner graph. Tanner graphs are a sub-class of factor graphs, so the sum-product algorithm can be directly applied to their normalized form. LDPC decoders were among the first to make explicit use of iterative message-passing decoding on factor graphs using the sum-product algorithm. Figure 2.5.10 shows Tanner graphs for a (3,4)-regular and an irregular LDPC code.

A decoder for an LDPC factor graph follows a procedure similar to the following:

1. All messages are initialized with a uniform probability mass.
2. Samples are received from the channel. The channel information is input to the graph at the variable nodes. Each input message is in the form of a probability mass. Set number of iterations to zero.
3. The equality nodes update their outgoing messages for each edge using the sum-product algorithm. The results are transmitted to the check nodes. During the first iteration, all outgoing messages are simply equal to the channel information.
4. The parity-check nodes update their outgoing messages using the sum-product algorithm. The results are transmitted back to the equality nodes. Increment the number of iterations.
5. After a "sufficient number" of iterations, sample the output messages. Make decisions on each variable based on maximum local probability, and stop.
6. Otherwise, return to step 3.

LDPC codes have a very simple structure which is easy to analyze. A variety of techniques have been developed to design and implement good LDPC codes. Using these techniques, codes have been found which come within 0.005dB of the Shannon limit for very large block lengths [20].

2.5.5 Block Turbo Codes.

Block Turbo Codes (BTCs), also known as Turbo Product Codes (TPCs) or Block Product Codes (BPCs), are two-dimensional constructions made from simple linear block codes. The term "Product code" describes the code's geometric construction. The term "Turbo" indicates the use of an iterative decoding algorithm. The IEEE 802.16a wireless communication standard refers to them as BTCs. For the purposes of this thesis, it is often easiest to refer to them simply as "Product codes."

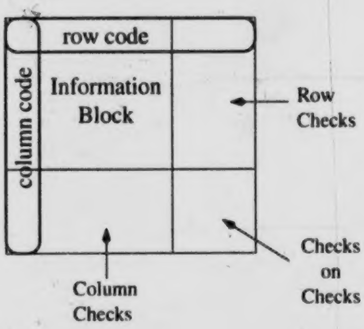


Figure 2.5.11: BTC codeword structure.

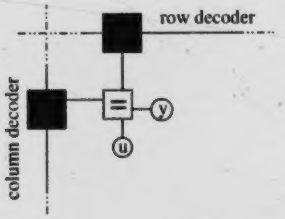


Figure 2.5.12: Concatenation of component decoders using equality nodes.

A BTC is defined by a (n_r, k_r) *row code* and a (n_c, k_c) *column code*. The information bits are arranged in a $k_r \times k_c$ rectangle. Each row is encoded by the row code, after which each column is encoded by the column code. The resulting rectangular codeword is illustrated in Figure 2.5.11.

Suppose the row code is described by factor graph $\mathcal{G}_r$, and the column code by $\mathcal{G}_c$. Then the product code's factor graph, $\mathcal{G}_{r\times c}$, contains n_c copies of $\mathcal{G}_r$ and n_r copies of $\mathcal{G}_c$. Each bit b in the codeword is shared by exactly one row code instance and exactly one column code instance. At the variable position corresponding to b, the two graphs are joined by an equality node, as shown in Figure 2.5.12.

There are many ways to draw the complete factor graph for a product code, and many ways to schedule message passing for sum-product decoding. A factor graph for an $(8,4)^2$ Hamming BTC is depicted in Figure 2.5.13. The row codes are labeled R1,..., R8 and the column codes are labeled C1,..., C8. We could also draw this factor graph by placing each $\mathcal{G}_r$ and $\mathcal{G}_c$ side-by-side in one long column on the left, with all the equal nodes in a column on the right. The resulting graph would

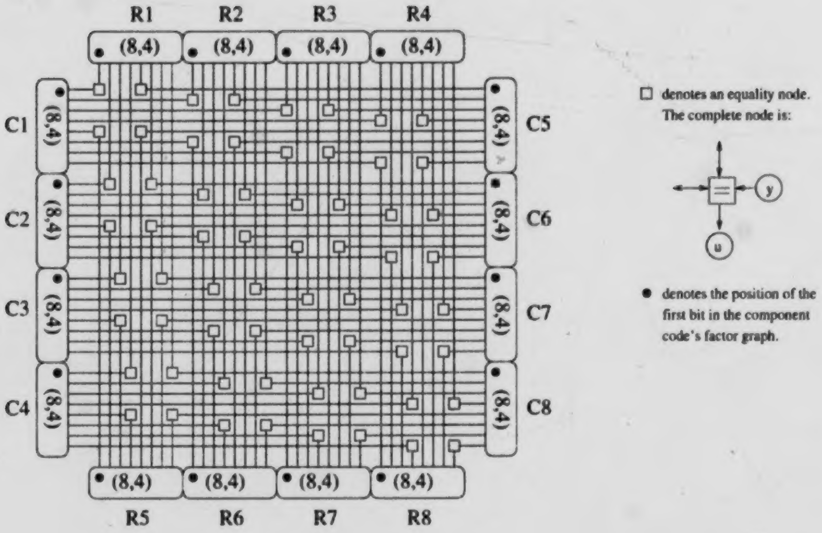

R1
R2
R3
R4
(8,4)
C1
C2
C3
C4
C5
C6
C7
C8
R5
R6
R7
R8
□ denotes an equality node. The complete node is:
=
y
u
● denotes the position of the first bit in the component code's factor graph.

Figure 2.5.13: Factor graph for an $(8, 4)^2$ Hamming BTC.

look almost like an LDPC code, with a very regular, structured interleaver.

A simple decoding algorithm for BTCs, based on the sum-product algorithm, is as follows [37]:

1. All internal messages are initialized with uniform probability masses.
2. Channel information is received. It is input to the equality nodes in the form of probability masses.
3. The equality nodes compute their output messages for each edge, and forward the results to all component decoders.
4. The component decoders perform local APP decoding based on the sum-product algorithm, and forward the results back to the equality nodes.
5. If a sufficient number of iterations has elapsed, sample the outputs from the equality nodes and stop.
6. Otherwise, return to step 3.

Because they are constructed from very simple component block decoders, BTC decoders are quite easy to construct. BTC's have been shown to approach the Shannon limit with very short block lengths compared to other Turbo-style codes (although BTCs usually have significantly higher rates).

One convenient feature of Product codes is their relatively simple geometric structure. If row and column component codes have minimum distance d_r and d_c, respectively, then the product code has minimum distance $d_r \cdot d_c$. As an example, extended Hamming codes have a minimum distance of four. The product of two Hamming codes therefore has a minimum distance of 16.

It is also very easy to enumerate the number of codewords which have minimum distance. The number of codewords of weight w in code C is often expressed as a function $A(w)$. This function is known as the distance spectrum for code C.

Knowing the distance spectrum for low-weight codewords allows us to estimate the performance of a MAP decoder. This estimate [13], called the minimum-distance approximation, is given by

$$P_e \approx \frac{d_{min}}{2n} \cdot A(d_{min}) \cdot \text{erfc}\left(\sqrt{\frac{R \cdot d_{min} \cdot E_b}{N_0}}\right). \qquad (2.5.1)$$

This approximation becomes increasingly accurate at high SNR, where minimum-distance errors dominate the probability.

Performance curves for some BTCs are shown in Figure 2.5.14. Each code uses identical Hamming codes for row and column components. The curves are derived from (2.5.1). Sum-product decoders for BTCs tend to perform very close to the MAP performance, so Figure 2.5.14 represents what we should expect to see from a good implementation. The performance of a BTC is not as close to Capacity as a Turbo code, but the BTC has a dramatically shorter block length and no error floor.

2.5.6 Summary

There are a variety of good codes, most of which are constructed by weaving together simple component codes. Turbo and LDPC codes provide performance very close to the Shannon limit, but may require enormous block lengths. Turbo codes also require careful and often complex design procedures to avoid error floors. Product Codes, by contrast, provide decent error correction with comparatively tiny system size. These codes are compared in Figure 2.5.15, in which their performance

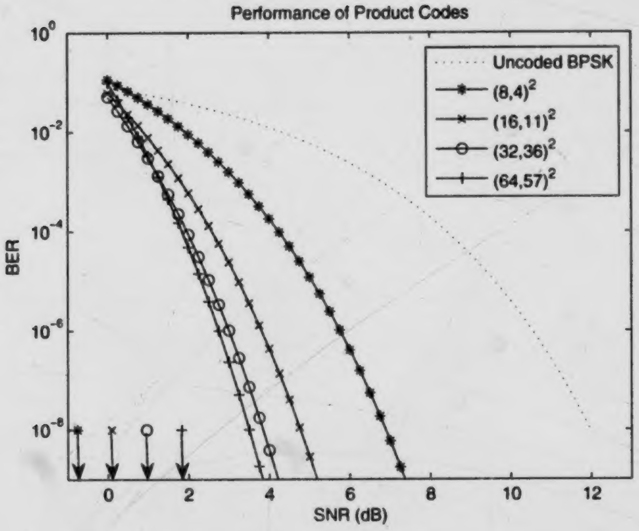


Figure 2.5.14: Performance of some Block Turbo Codes. The BPSK limits are indicated by arrows for each code rate [37].

is plotted alongside the BPSK limit. The data points represent the SNR at which each code attains a BER of 10^{-5}. The performance of Hamming block codes is shown alongside Product Codes constructed from Hamming components.

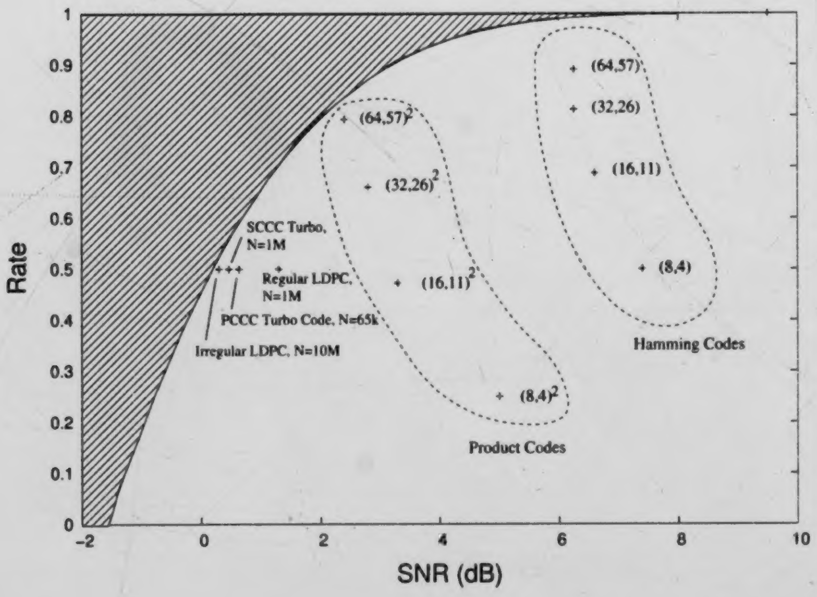


Figure 2.5.15: Performance of various codes relative to the BPSK limit. Data points represent the SNR needed to acieve a BER of 10^{-5}. Code lengths are also indicated for regular LDPC [69], irregular LDPC [20], PCCC Turbo [15], and SCCC Turbo codes [81].

Chapter 3

Trellis Codes

3.1 Trellis graphs for block and convolutional codes.

A *trellis* is a graph which is usually thought to represent a convolutional code. In a convolutional code, encoding is performed by something like a FIR filter, rather than through explicit matrix multiplication as in (2.3.1). In practical terms, though, all information messages end after some number of symbols.

It is common practice to *terminate* convolutionally coded messages after a fixed number of information symbols. When such terminations are imposed, a message consists of k information symbols and n coded symbols, and the convolutional code is, in effect, a block code. Convolutional codes are also typically linear, so all the concepts of Section 2.3 can be applied. We will only discuss the ways in which a trellis represents a linear block code.

The trellis may loosely be thought of as a Markov model of the encoding device. A trellis consists of *states* and labeled *branches*. The states are arranged in columns, and each column $\mathcal{S}_i$ represents the possible states of the encoder at a particular time i. Branches connect the states in $\mathcal{S}_i$ to those in $\mathcal{S}_{i+1}$, and represent the allowable transitions of the encoder's state.

A *branch* is defined by the ordered triple $b_{qr} = \left(s_q, l_{qr}, s'_r\right)$, where $s_q \in \mathcal{S}_i$ and $s'_r \in \mathcal{S}_{i+1}$. The *label* l_{qr} is a sequence of one or more channel symbols. Each branch may also be associated with a sequence of one or more information symbols. The information symbols are interpreted as the input to the encoding device, and the channel symbols are the output produced by the encoder.

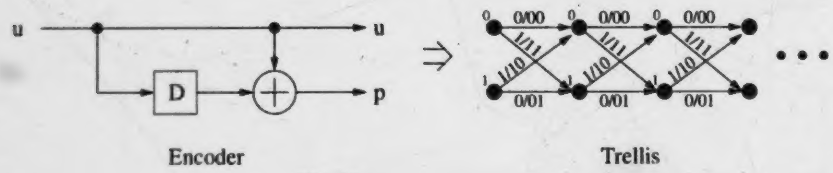


Figure 3.1.1: A simple two-state convolutional code.

Example 3.1.1. : An encoder and its corresponding trellis for a two-state convolutional code are shown in Figure 3.1.1. The encoder takes a single bit u as input, and outputs a pair of bits (u, p). The box labeled D represents a delay (i.e. a one-bit shift-register). Thus $p = u_i \oplus u_{i-1}$, where i is the time-index and $\oplus$ is modulo-2 addition. This code's trellis has two states, which correspond to the value stored in the shift-register. The branches are labeled with u/up, representing the input and output of the encoder.

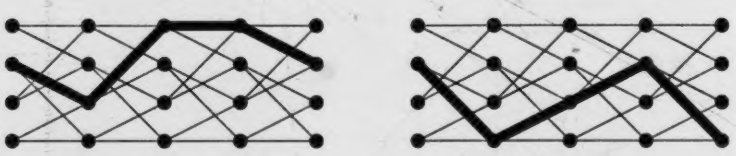
□

Let $\mathcal{L}_i$ refer to the set of branches which connect states in $\mathcal{S}_i$ to those in $\mathcal{S}_{i+1}$. A *trellis section* $\mathcal{T}_i$ for time i is defined by the ordered triple $\mathcal{T}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{S}_{i+1})$. For clarity, we distinguish between the *state variables* $\mathcal{S}_i$ and the *state alphabet* $\mathcal{A}_s^{(i)}$. To say that $\mathcal{A}_s^{(j)} = \mathcal{A}_s^{(i)}$ is to say that the trellis has the same set of possible states at time i and at time j. To say that $\mathcal{S}_i = \mathcal{S}_j$ is to say that the encoder must be in *exactly* the same state at time i and at time j.

A trellis must have some *termination*, which is a constraint on the initial and final states. A *path* is a connected sequence of branches which traverses the entire trellis and satisfies the termination condition. The *conventional* termination imposes, for a trellis with L sections, the condition that $\mathcal{S}_0$ and $\mathcal{S}_L$ each have a single state (i.e. the encoder begins and finishes in a known state).

The *tailbiting* termination requires that $\mathcal{S}_i = \mathcal{S}_{i+L}$, meaning the encoder must start and end *in the same state*. A tailbiting trellis is thus a circular structure in which every path must revisit itself after L transitions. The paths in a trellis represent the *codewords* in the corresponding block code.

In a tailbiting trellis, there are connected sequences of branches, called *pseudocodewords*, which do not satisfy the termination condition. A pseudocodewords



Valid Codeword

Pseudocodeword

Figure 3.1.2: Valid and invalid paths in a tailbiting trellis with four sections.

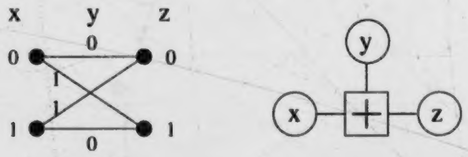


Figure 3.1.3: Trellis-style constraint and factor-graph for $f(x,y,z) = x+y+z = 0$.

does not terminate in the same state as it began. A pseudocodeword and a valid codeword are contrasted in the tailbiting trellis shown in Figure 3.1.2.

Each trellis section represents the functional relationship between variables $\mathcal{L}_i$, $\mathcal{S}_i$, and $\mathcal{S}_{i+1}$. When these variables are binary, the trellis section's constraint can be represented in terms of $\mathbb{Z}_2$ operations. An example of this is shown in Figure 3.1.3, where '$+$' denotes $\mathbb{Z}_2$ addition. Thus while factor graphs represent the relationships between component functions and variables, a trellis section represents the actual constraint imposed at a function node.

The state variables in a trellis are hidden variables, and the labels $\mathcal{L}_i$ are channel symbol variables. It is therefore straightforward to draw and interpret the factor graph of a trellis.

Example 3.1.2. A tailbiting trellis for the (8,4) Hamming code [19] is shown in Figure 3.1.4, and a corresponding factor graph is shown in Figure 3.1.5. Each branch in the trellis is labeled with a single information bit, U_i, and a pair of channel bits, L_i. The trellis thereby expresses functional relationships between $\underline{u}$, $\underline{x}$, and the hidden state variables $\mathcal{S}_i$. The state variables are indicated in Figure 3.1.5 by double-circles.

□

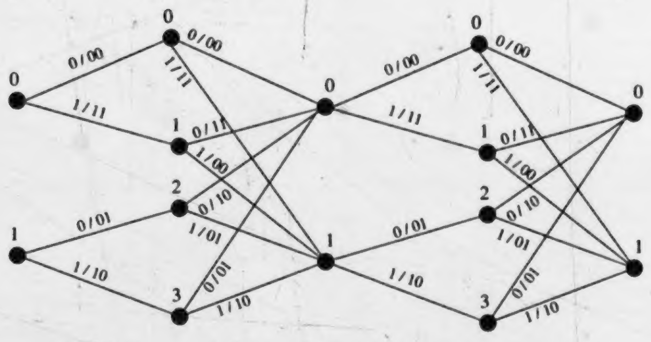

0
0/00
1/11
1
0/01
1/10
0
1
2
3
0/00
1/11
0/11
1/00
0/10
1/01
0/01
1/10
0
1
0/00
1/11
0/01
1/10
0
1
2
3
0/00
1/11
0/11
1/00
0/10
1/01
0/01
1/10
0
1

Figure 3.1.4: Tailbiting trellis for the (8,4) Hamming code.

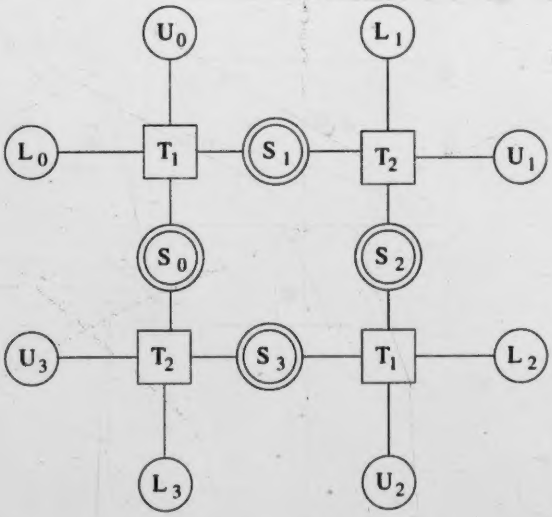

U_0
L_1
L_0
T_1
S_1
T_2
U_1
S_0
S_2
U_3
T_2
S_3
T_1
L_2
L_3
U_2

Figure 3.1.5: Factor graph corresponding to the tailbiting trellis of Figure 3.1.4.

3.2 Soft-information trellis decoding.

The BCJR algorithm, named for the authors who introduced it [9], calculates the exact trellis state probabilities when applied to conventional trellises. When applied to tailbiting trellises, it produces approximate state probabilities which are corrupted by the influence of pseudocodewords. Good results are nevertheless obtainable with tailbiting trellises.

The BCJR algorithm has also been shown to be an instance of the sum-product algorithm when the trellis is represented as a factor graph. From this perspective, a conventional trellis is a tree, and therefore the sum-product algorithm is equivalent to maximum a posteriori (MAP) decoding. In the tailbiting case, the graph has one large cycle. Pseudocodewords can therefore be described in terms of loopy graphs, as discussed in Section 2.4.2.

3.2.1 MAP decoding on trellises.

Before explaining the BCJR algorithm, we reintroduce *maximum a posteriori* decoding in the context of trellises, to establish some necessary concepts and notation. MAP decoding for block codes is discussed in Section 2.4.1.

We assume that the encoder traverses a complete trellis path, and that its output is transmitted over some communications channel, beyond which lies a decoder. The decoder attempts to infer the sequence of information symbols based on observations of the channel's output. Let the actual information symbol at time i be written $\mathbf{u}_i$, and let the actual channel symbol at time i be written $\mathbf{x}_i$. Further let $\mathbf{p}$ refer to the actual path of trellis transitions traversed by the encoder, and let $\mathbf{s}_i$ refer to the actual trellis state after symbol i. To the receiver, the information bits, channel symbols, path and states are random variables.

The channel's output is a sequence of random samples, denoted r_i. It is assumed that the channel is memoryless, so that r_i depends only on x_i, and that estimates are available for

$$\lambda_i^{(j)} = P\left(r_i \,|\, \mathbf{x}_i = x_i^{(j)}\right) \tag{3.2.1}$$

for every r_i and $x_i^{(j)}$, which $x_i^{(j)}$ refers to the j^{th} member of the symbol alphabet $\mathcal{A}_x$.

The *maximum a posteriori* (MAP) decoding solution for $\mathbf{x}_i$, written $\widehat{x}_i$, is that $u_i^{(j)}$ which maximizes

$$\rho_i^{(j)} = P\left(\mathbf{u}_i = u_i^{(j)} \mid r_i\right) \tag{3.2.2}$$

A closely related problem is to find the conditional probabilities of states in the trellis. Let $\underline{r}$ refer to the complete sequence of channel observations $\langle r_1, r_2, \ldots, r_L \rangle$. The trellis state probability $\sigma_i^{(j)}$ for the j^{th} state at the i^{th} time instant is defined as

$$\sigma_i^{(j)} = P\left(\mathbf{s}_i = s_i^{(j)} \mid \underline{r}\right) \tag{3.2.3}$$

The *a posteriori* information symbol probabilities can generally be inferred from the trellis state probabilities.

3.2.2 Transition matrices

A trellis section can be thought of as a Markov model, and as such the evolution of conditional probabilities can be determined using the Markov transition matrix. The Markov transition matrix for a trellis section $\mathcal{T}_i$ is an $n \times m$ matrix Γ_i whose entries are

$$\gamma_{jk} = P\left(\mathbf{s}_{i+1} = s_{i+1}^{(k)} \mid \mathbf{s}_i = s_i^{j}\right) \tag{3.2.4}$$

where $n = |\mathcal{S}_i|$ and $m = |\mathcal{S}_{i+1}|$. If the branch b_{jk} does not exist, then $\gamma_{jk} = 0$.

Let $\underline{\sigma}_i$ be a row vector in which the j^{th} element is equal to the trellis state probability $\sigma_i^{(j)}$. If $\underline{\sigma}_i$ and Γ_i are known, then the probability mass $\underline{\sigma}_{i+1}$ is deduced from the rule

$$\underline{\sigma}_{i+1} = \underline{\sigma}_i \cdot \Gamma_i \tag{3.2.5}$$

We are often interested in merging two trellis sections. Let $\mathcal{T}_i = (\mathcal{S}_i, \mathcal{L}_i, \mathcal{S}_{i+1})$ and $\mathcal{T}_{i+1} = (\mathcal{S}_{i+1}, \mathcal{L}_{i+1}, \mathcal{S}_{i+2})$ be two trellis sections. We define the *trellis section product* '⊗' as

$$\mathcal{T}_i' = \mathcal{T}_i \otimes \mathcal{T}_{i+1} = \left(\mathcal{S}_i, \mathcal{L}_i', \mathcal{S}_{i+2}\right) \tag{3.2.6}$$

where $\mathcal{L}_i'$ is the set of branches which connect states in $\mathcal{S}_i$ to those in $\mathcal{S}_{i+2}$. A branch b_{qs}' is in $\mathcal{L}_i'$ if there exist branches $b_{qr} \in \mathcal{L}_i$ and $b_{rs} \in \mathcal{L}_{i+1}$, where q, r, and s are state indices. The label l' associated with b_{qs}' is the ordered pair $\left(l_{qr}, l_{rs}\right)$.

The transition matrix for $\mathcal{T}_i'$ is

$$\Gamma_i' = \Gamma_i \times \Gamma_{i+1} \tag{3.2.7}$$

3.2.3 Details of the BCJR algorithm.

The algorithm consists of two phases, forward and backward. For the forward phase, each $\mathcal{S}_i$ is assigned a *forward probability mass* $\underline{\alpha}_i$, which is a row vector, and is conditioned only on channel observations prior to time i. The initial vector for a conventional trellis is $\underline{\alpha}_0 = \langle 1 \rangle$.

The transition matrices are populated with branch probabilities

$$\begin{aligned} \gamma_{jk} &= P\left(s_k^{(i+1)}, y_i \,|\, s_j^{(i)}\right) && (3.2.8) \\ &= \lambda_i^{(jk)} \cdot P\left(u_{jk}\right) && (3.2.9) \end{aligned}$$

where $P\left(u_{jk}\right)$ is the *a priori* probability of the information symbol u_{jk}.

The forward phase is carried out by iteration of the rule

$$\underline{\alpha}_{i+1} = \underline{\alpha}_i \cdot \Gamma_i \tag{3.2.10}$$

Similarly, each $\mathcal{S}_i$ is assigned a *backward probability mass* $\underline{\beta}_i$, which is conditioned on channel observations at times greater than or equal to i. The backward phase is carried out using a rule similar to (3.2.10):

$$\underline{\beta}_i = \underline{\beta}_{i+1} \cdot \Gamma_i^{\mathrm{T}} \tag{3.2.11}$$

The trellis state probabilities are obtained by the product

$$\underline{\sigma}_i = \underline{\alpha}_i \odot \underline{\beta}_i \tag{3.2.12}$$

where the '$\odot$' product refers to element-wise multiplication.

For tailbiting trellises, the algorithm is slightly more complicated, and fails to produce the exact trellis state probabilities. Because $|\mathcal{S}_0| > 0$, the initial forward probability mass is not known (i.e. $\underline{\alpha}_0 \neq \langle 1 \rangle$). Usually for tailbiting trellises, $\underline{\alpha}_0$ is set to a uniform distribution, or some other similarly arbitrary distribution. The forward and backward phases continue around the trellis more than once, so that successive estimates of $\underline{\alpha}_0$ are obtained.

Let $A_0 = \prod_{i=1}^{L} \Gamma_i$. In the tailbiting BCJR algorithm, let $\underline{\alpha}_0^{(k)}$ refer to the estimate $\underline{\alpha}_0$ after k passes around the trellis. As k increases, $\underline{\alpha}_0^{(k)}$ converges to an eigenvector of A_0. Similarly, $\underline{\beta}_0^{(k)}$ converges to an eigenvector of A_0^{T}. These eigenvectors are not the exact probability solutions. The inexactness of the tailbiting BCJR algorithm is due to the influence of pseudocodewords.

3.2.4 An example decoder: (8,4) tailbiting Hamming trellis.

To fully illustrate the BCJR algorithm, we will now examine a complete decoder for the tailbiting (8,4) Hamming trellis shown in Figure 3.1.4. Examination of the trellis reveals that there are only two sections with unique constraints. The corresponding function nodes are labeled T_1 and T_2 in the factor graph of Figure 3.1.5.

The first step in designing a sum-product decoder is to specify the code's normal graph. As described in Section 2.3.1, the normal graph should include every input and output variable, while hidden variables are omitted. The normal graph for the (8,4) Hamming code is shown in Figure 3.2.1. Note that the "L_i" variable nodes, which represent *pairs* of channel bits in Figure 3.1.5, have been replaced by the more explicit "Tree" function node in Figure 3.2.1. The variables "x_j" are the individual channel bits.

The next step in decoder design is to label all of the messages in the graph, and to identify the direction and degree of all messages. We refer to this as the *message schematic*, shown in Figure 3.2.3. The large boxes labeled 1, ..., 4 represent the implementations of each trellis section. Boxes 1 and 3 are implementations of T_1, and boxes 2 and 4 are implementations of T_2. The triangle represents a comparator.

The messages labeled $\underline{\lambda}_j$ represent the channel information. Each $\underline{\lambda}_j$ is a binary

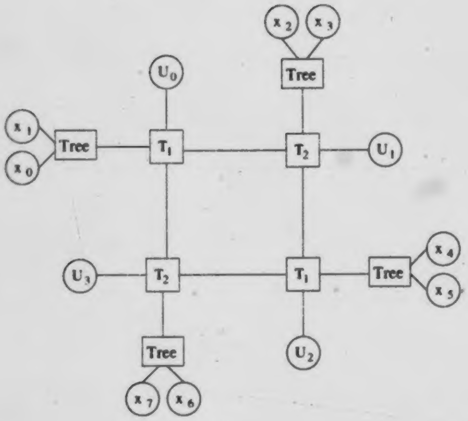

x_2
x_3
Tree
U_0
x_1
Tree
T_1
T_2
U_1
x_0
U_3
T_2
T_1
Tree
x_4
x_5
Tree
U_2
x_7
x_6

Figure 3.2.1: Normal graph for (8,4) Hamming code.

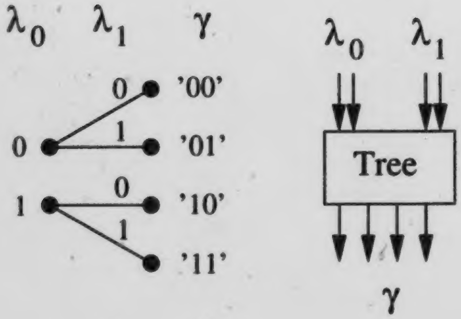

λ_0
λ_1
γ
0
'00'
1
0
'01'
0
1
'10'
1
'11'
λ_0
λ_1
Tree
γ

Figure 3.2.2: Illustration of the "Tree" function node.

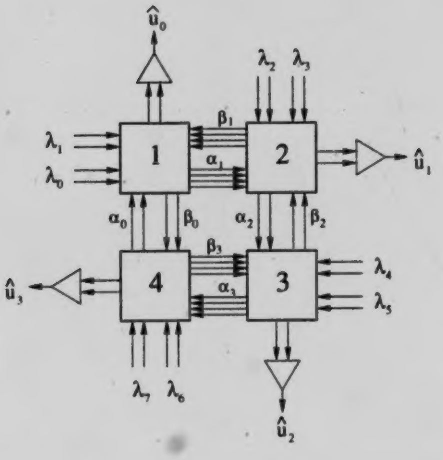


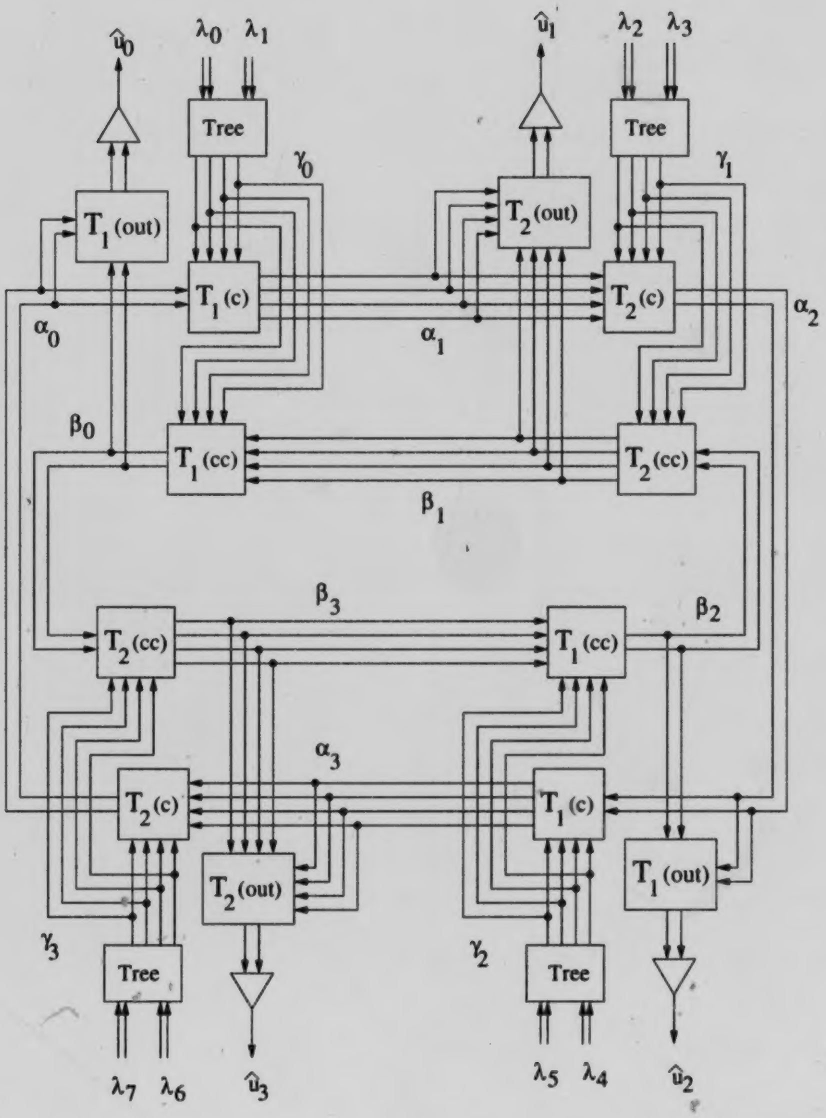
Figure 3.2.3: Message schematic diagram for the (8,4) Hamming decoder.

probability mass (a column vector with two members). If r_j is the measurement for the j^{th} bit of the channel's output, then

$$\underline{\lambda}_j \equiv \begin{bmatrix} P\left(\mathbf{x_j}=0\,|\,r_j\right) \\ P\left(\mathbf{x_j}=1\,|\,r_j\right) \end{bmatrix} \tag{3.2.13}$$

There are two sets of internal messages in the network, labeled α_i and β_i. These represent separate estimates of the probability masses for hidden state nodes S_i. The $\underline{\alpha}$ messages propagate around the graph in the clockwise direction, while the $\underline{\beta}$ messages propagate counter-clockwise. The decoder's final digital output is produced by a comparator, whose decisions are the information bit estimates $\widehat{u}_i$.

The next step in decoder design is to characterize the local sum-product operations for each node. To do this, we first need a more fine-grained *node schematic*, which identifies the actual sum-product components. The complete decoder schematic is shown in Figure 3.2.4, which reveals that eight distinct components must be implemented to complete the decoder. These include T_1 (c), T_1 $(cc)$and T_1 (out), which implement propagation through T_1in the clockwise, counterclockwise and outward directions, respectively. We need to implement the same components for node T_2. The *Tree* node only needs implementation in one direction, as information is never fed back into the channel. Lastly, a comparator is required to make the final bit decisions.



$\hat{u}_0$
λ_0
λ_1
$\hat{u}_1$
λ_2
λ_3
Tree
Tree
γ_0
γ_1
T_1 (out)
T_2 (out)
T_1 (c)
T_2 (c)
α_0
α_1
α_2
β_0
T_1 (cc)
T_2 (cc)
β_1
β_3
β_2
T_2 (cc)
T_1 (cc)
α_3
T_2 (c)
T_1 (c)
T_2 (out)
T_1 (out)
γ_3
γ_2
Tree
Tree
λ_7
λ_6
$\hat{u}_3$
λ_5
λ_4
$\hat{u}_2$

Figure 3.2.4: Complete message and node schematic for (8,4) Hamming decoder.

We begin with the *Tree* node, illustrated in Figure 3.2.2. For a three-edge node such as this one, the sum-product rule (2.4.3) can be expressed as a matrix multiplication: $\underline{\gamma} = Tree(\underline{\lambda}_1) \cdot \underline{\lambda}_0$, where $\underline{\gamma}$, $\underline{\lambda}_0$ and $\underline{\lambda}_1$ are probability-mass vectors. It is easily verified that the matrix $Tree(\underline{\lambda}_1)$ has the form

$$\begin{bmatrix} \gamma^{(00)} \\ \gamma^{(01)} \\ \gamma^{(10)} \\ \gamma^{(11)} \end{bmatrix} = \begin{bmatrix} \lambda_1^{(0)} & \\ \lambda_1^{(1)} & \\ & \lambda_1^{(0)} \\ & \lambda_1^{(1)} \end{bmatrix} \cdot \begin{bmatrix} \lambda_0^{(0)} \\ \lambda_0^{(1)} \end{bmatrix} \tag{3.2.14}$$

$$\underline{\gamma} = Tree(\underline{\lambda}_1) \cdot \underline{\lambda}_0. \tag{3.2.15}$$

The $Tree(\underline{\lambda}_1)$ matrix of (3.2.14) is directly derived from the trellis section of Figure 3.2.2. The columns of $Tree(\underline{\lambda}_1)$ are indexed by the left states, and the rows are indexed by the right states of the trellis section. If a branch exists between state b on the left and state c on the right, and that branch has label x, then the bc^{th} position of $Tree(\underline{\lambda}_1)$ contains $\lambda_1^{(x)}$. If no branch exists, then the bc^{th} position contains a zero.

The matrices for $T_i(c)$ and $T_i(cc)$ are derived from their trellises in similar manner, resulting in

$$T_1(c) = \begin{bmatrix} \gamma^{(00)} & 0 \\ \gamma^{(11)} & 0 \\ 0 & \gamma^{(01)} \\ 0 & \gamma^{(10)} \end{bmatrix} \qquad T_2(c) = \begin{bmatrix} \gamma^{(00)} & \gamma^{(11)} & \gamma^{(10)} & \gamma^{(01)} \\ \gamma^{(11)} & \gamma^{(00)} & \gamma^{(01)} & \gamma^{(10)} \end{bmatrix} \tag{3.2.16}$$

$$T_1(cc) = [T_1(c)]^{\mathrm{T}} \qquad T_2(cc) = [T_2(c)]^{\mathrm{T}}$$

The propagation rules for messages $\underline{\alpha}$ and $\underline{\beta}$ are

$$\underline{\alpha}_{i+1} = T_i \cdot \underline{\alpha}_i \tag{3.2.17}$$

$$\underline{\beta}_i = T_i^{\mathrm{T}} \cdot \underline{\beta}_{i+1} \tag{3.2.18}$$

The $T_1(out)$ matrix is slightly more complicated. We want to extract the *information bit* probabilities. For any section in the (8, 4) Hamming trellis of Figure 3.1.4, the information symbol u induces a partition of the *right states* of the section. The even-numbered states correspond to $u = 0$, and the odd-numbered states correspond to $u = 1$. We may therefore express a constraint between u and the right state variable S. There are several equivalent ways to represent this relationship in

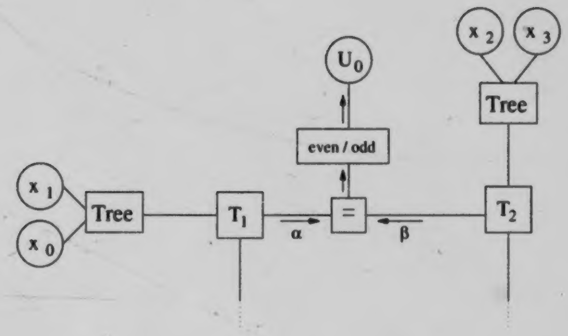


Figure 3.2.5: Illustration of constraint between information bit and state variable (normal form).

the factor graph, one of which is shown in Figure 3.2.5. Implementation of (2.4.3) for this constraint consists of the Kronecker product $\underline{\alpha} \odot \underline{\beta}$, followed by separate summation over the odd and even members. In matrix form, this can be written as

$$T_1\,(out) = \begin{bmatrix} \beta^{(0)} & 0 \\ 0 & \beta^{(1)} \end{bmatrix} \qquad T_2\,(out) = \begin{bmatrix} \beta^{(0)} & 0 & \beta^{(2)} & 0 \\ 0 & \beta^{(1)} & 0 & \beta^{(3)} \end{bmatrix} \tag{3.2.19}$$

with the propagation rules

$$\begin{array}{ll} u_0 = \mathcal{D}\{[T_1\,(out)] \cdot \underline{\alpha}_0\} & u_1 = \mathcal{D}\{[T_2\,(out)] \cdot \underline{\alpha}_1\} \\ u_2 = \mathcal{D}\{[T_1\,(out)] \cdot \underline{\alpha}_2\} & u_3 = \mathcal{D}\{[T_2\,(out)] \cdot \underline{\alpha}_3\} \end{array} \tag{3.2.20}$$

where $\mathcal{D}$ denotes the decision operation (less-than or greater-than).

3.3 Construction of block code trellises.

3.3.1 Generator matrices and the trellis product.

A trellis can be constructed from a linear block code's generator matrix. To do this, an *elementary subtrellis* is first constructed for each row in the generator matrix. The subtrellises are then combined to form the complete trellis graph using a *trellis product* operation [49].

A generator matrix G consists of rows g_i. The *span* of a row g_i is defined as a contiguous range of indices which covers all non-zero elements of g_i. In a conventional trellis construction, the span begins at the left-most position j for which $g_{i,j} \neq 0$. The span ends at the right-most position k for which $g_{i,k} \neq 0$. The row g_i is said to be *active* in the range $[j, k-1]$.

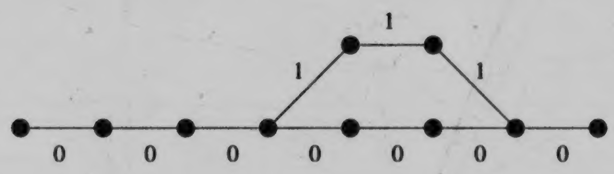


Figure 3.3.1: An example row subtrellis.

An *elementary subtrellis*, consisting of two paths, is constructed for each row as follows. The first path is the zero sequence. The second path diverges from the zero sequence at position j, and merges again with the zero sequence at position k. The second path is labeled with the entries from g_i.

Example 3.3.1. A generator matrix G is shown with row spans in bold, and active positions underlined. The subtrellis corresponding to row g_4 is shown in Figure 3.3.1.

$$G = \begin{bmatrix} \underline{\mathbf{1}} & \underline{\mathbf{0}} & \underline{\mathbf{0}} & \underline{\mathbf{0}} & \underline{\mathbf{1}} & \underline{\mathbf{1}} & \mathbf{1} \\ 0 & \underline{\mathbf{1}} & \underline{\mathbf{0}} & \underline{\mathbf{0}} & \underline{\mathbf{1}} & \underline{\mathbf{0}} & \mathbf{1} \\ 0 & 0 & \underline{\mathbf{1}} & \underline{\mathbf{0}} & \underline{\mathbf{0}} & \underline{\mathbf{1}} & \mathbf{1} \\ 0 & 0 & 0 & \underline{\mathbf{1}} & \underline{\mathbf{1}} & \mathbf{1} & 0 \end{bmatrix}$$

□

Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be the elementary subtrellises for two rows, g_1 and g_2, respectively. The product of two elementary trellises $\mathcal{R}_P = \mathcal{R}_1 \odot \mathcal{R}_2$ may be taken one section at a time. To simplify the discussion, we now omit certain indices and describe the states and branches of $\mathcal{R}_1$ using Roman letters, whereas those of $\mathcal{R}_2$ are written with Greek letters.

Let $b = (s, l, s')$ be a branch in section $\mathcal{T}_{1i}$ of subtrellis $\mathcal{R}_1$, and let $\beta = (\sigma, \lambda, \sigma')$ be a branch in section $\mathcal{T}_{2i}$ of subtrellis $\mathcal{R}_2$. To construct the corresponding section $\mathcal{T}_i$ of $\mathcal{R}_P$, we take the product $p = b \odot \beta$ for every $b \in \mathcal{T}_{1i}$ and $\beta \in \mathcal{T}_{2i}$. The left-state of p is the ordered pair (s, σ), the right-state is the ordered pair (s', σ'). The label for p is equal to $l \oplus \lambda$, where $\oplus$ represents the group product associated with the branch labels. Typically $\oplus$ is vector addition modulo two.

After taking the product for all branches in the corresponding elementary subtrellis sections, we arrive at a trellis section with $|S_{1i}| \cdot |S_{2i}|$ left-states and $\left|S_{1(i+1)}\right| \cdot$

$\left|S_{2(i+1)}\right|$ right-states. Taking the products over all branches for all sections in the two trellises completes the product. By recursively applying this product to all elementary subtrellises, the complete trellis is constructed.

Example 3.3.2. The generator matrix for an (8,4) Hamming code is

$$G_8 = \begin{bmatrix} \underline{\mathbf{11}} & \mathbf{11} & 00 & 00 \\ \underline{\mathbf{01}} & \underline{\mathbf{01}} & \underline{\mathbf{10}} & \mathbf{10} \\ 00 & \underline{\mathbf{11}} & \mathbf{11} & 00 \\ 00 & 00 & \underline{\mathbf{11}} & \mathbf{11} \end{bmatrix}$$

Note that the columns have been arranged in pairs. We will construct a trellis with four sections, in which each branch is labeled with a pair of bits. Let a_i refer to the number of active (underlined) rows in column i, $0 \leq i \leq 3$. In the final trellis for this code, the number of states $|S_{i+1}| = 2^{a_i}$. We will therefore arrive at a trellis with the state profile $1,4,4,4,1$.

The elementary subtrellises for this code are shown in Figure 3.3.2. The product of the first two subtrellises is carried out in Figure 3.3.3. After applying the procedure to every row, we arrive at the complete trellis shown in Figure 3.3.4. To simplify the diagram, labels are indicated by the line-style of branches. The mapping between labels and line-styles is indicated in the first trellis section.

☐

3.3.2 Construction of tailbiting trellises.

A tailbiting trellis for a block code can also be constructed using the trellis product [19]. From the tailbiting perspective, the time index is periodic modulo L. We therefore define the row span strictly as a contiguous range of indices which covers all non-zero entries in the row. Because the indices are periodic, the span may "wrap around" from the left to the right of the generator matrix. This is commonly referred to as a *circular span*.

Example 3.3.3. A generator matrix for an (8,4) Hamming code is shown below, illustrating the use of a circular span.

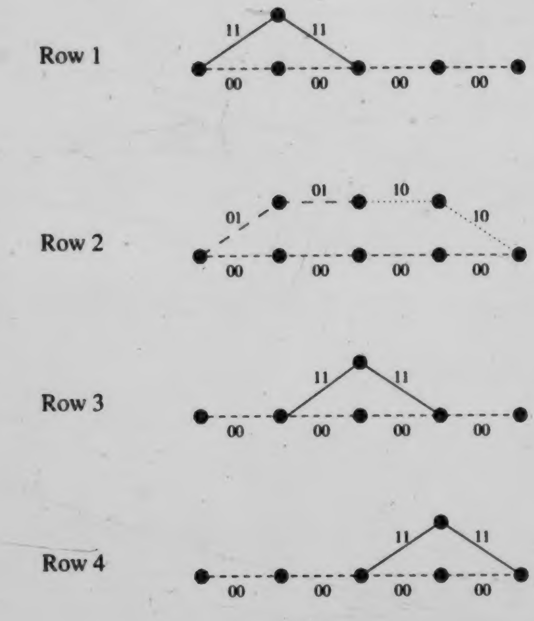

Row 1
11
11
00
00
00
00
Row 2
01
01
10
10
00
00
00
00
Row 3
11
11
00
00
00
00
Row 4
11
11
00
00
00
00

Figure 3.3.2: Elementary subtrellises for a conventional (8,4) Hamming trellis.

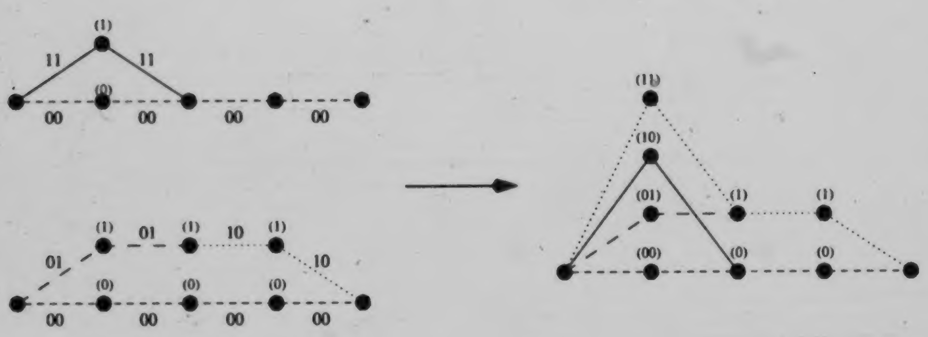

(1)
11
11
(0)
00
00
00
00
(1)
01
(1)
10
(1)
01
10
(0)
(0)
(0)
00
00
00
00
(11)
(10)
(01)
(1)
(1)
(00)
(0)
(0)

Figure 3.3.3: Trellis product of rows one and two.

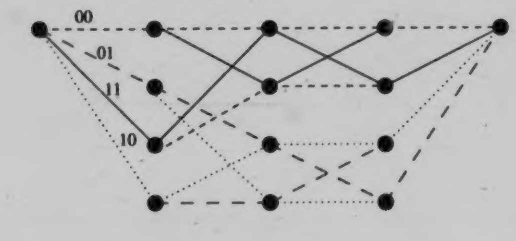


Figure 3.3.4: Complete trellis after all row products are taken.

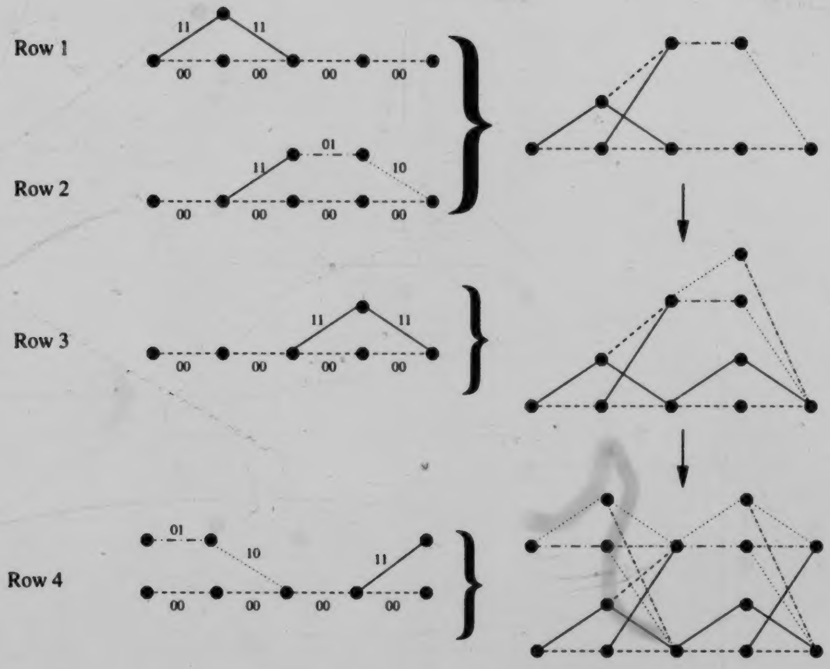


Figure 3.3.5: Construction of a tailbiting (8,4) Hamming code trellis.

$$G_8 = \begin{bmatrix} \underline{11} & 11 & 00 & 00 \\ 00 & \underline{11} & \underline{01} & 10 \\ 00 & 00 & \underline{11} & 11 \\ \underline{01} & 10 & 00 & \underline{11} \end{bmatrix}$$

The elementary subtrellises for this generator matrix are shown in Figure 3.3.5, which also illustrates the step-by-step construction procedure.

□

3.3.3 Squaring construction for Hamming and Reed-Muller codes.

In this section we describe a method for constructing the well-known Reed-Muller (RM) codes by "gluing together" a set of smaller codes. This method produces very compact trellises. The Hamming codes are also a subset of the RM codes.

We present an abbreviated summary of the squaring construction. For more detailed information, the reader is referred to [30, 22], and to the tutorial discussion in Chapter 5 of [73].

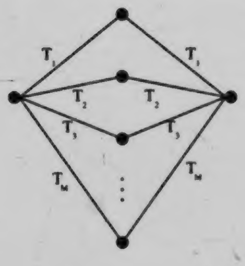


Figure 3.3.6: Illustration of the squaring construction.

RM codes are defined by two parameters, ρ and ν. An RM(ρ,ν) code has length $n = 2^{\nu}$ and redundancy $r = 2^{\nu-\rho} - 1$. The information-block length is $k = n - r$. RM(ν,ν) is the set of all length-n binary vectors, written $\mathbb{Z}_n$. RM$(-1,\nu)$ consists of the zero vector of length n.

An RM(ρ,ν) may be partitioned into a set of cosets, each of which is an RM$(\rho-1,\nu)$ code. Such a partition is denoted RM(ρ,ν)/RM$(\rho-1,\nu)$. Such a partition can be used to construct a code RM$(\rho,\nu+1)$ using a *squaring* operation, written

$$\mathrm{RM}(\rho,\nu+1) = |\mathrm{RM}(\rho,\nu)/\mathrm{RM}(\rho-1,\nu)|^2. \tag{3.3.1}$$

In general terms, the partition T/V consists of several cosets T_j. The squaring operation constructs a space $S = |T/V|^2$ which consists of all pairs (s_1, s_2) such that both s_1 and s_2 are in the same coset T_j. This construction is illustrated in Figure 3.3.6, in which each branch denotes an entire coset T_j.

We now assume that S, T, and V are linear spaces. Let C be a matrix whose rows form a basis for V, and define Q as the matrix whose rows are all of the distinct non-zero, linearly independent *coset selectors* for the partition T/V. We say that a row q_i is a coset selector if $T_j + q_i$ is a coset in T/V, where the sum $T_j + q_i$ indicates the addition of all members of T_j by q_i.

We may now write a matrix whose rows form a basis for S as

$$G_{S1} = \begin{bmatrix} C & \\ & C \\ Q & Q \end{bmatrix}. \tag{3.3.2}$$

If S, T and V are all RM codes, then (3.3.2) is a generator matrix for the constructed code.

For a given coset T_j in Figure 3.3.6, all vectors in T_j are labeled together on a single branch. A branch with multiple labels is said to contain *parallel branches*. The rows containing the C matrices in (3.3.2) are represented in the trellis as parallel branches. As a result, only the submatrix $\begin{bmatrix} Q & Q \end{bmatrix}$ is used for the trellis product construction.

A more powerful form of the squaring construction is the *two-level squaring* operation. This method uses a two-level partition $T/U/V$ to construct S. Again let C be the matrix whose rows form a basis for V. Let Q be the coset selector matrix for the partition U/V, and let Q_2 be the coset selector matrix for the partition T/U. Then the basis matrix for S is

$$G_{S2} = \begin{bmatrix} C & & & \\ & C & & \\ & & C & \\ & & & C \\ Q_2 & Q_2 & Q_2 & Q_2 \\ Q & Q & & \\ & Q & Q & \\ & & Q & Q \end{bmatrix}. \tag{3.3.3}$$

The occurrences of C in G_{S2} again induce parallel branches, and can be ignored. The row $\begin{bmatrix} Q_2 & Q_2 & Q_2 & Q_2 \end{bmatrix}$ induces a set of disjoint subtrellises, which are connected only at the initial and final states. The subtrellises all have the same form, which is constructed using the trellis product procedure, as shown in Figure 3.3.7. Figure 3.3.8 shows how the disjoint subtrellises are joined to form the complete trellis.

The two-level squaring procedure is indicated using the notation $S = |T/U/V|^4$. The method can be used to construct RM codes based on the relationship

$$\mathrm{RM}(\rho, \nu+2) = |\mathrm{RM}(\rho, \nu)/\mathrm{RM}(\rho-1, \nu)/\mathrm{RM}(\rho-2, \nu+2)|^4. \tag{3.3.4}$$

The parameters of various RM codes and their relative partition degrees are shown in Table 3.1, which is reproduced from [73].

3.3.4 Example: (16,11) Hamming code construction.

The extended Hamming codes are the set of $\mathrm{RM}(\nu-2, \nu)$ codes. The $\mathrm{RM}(2,4)$ code is therefore a (16,11) Hamming code. This code and its trellis can be con-

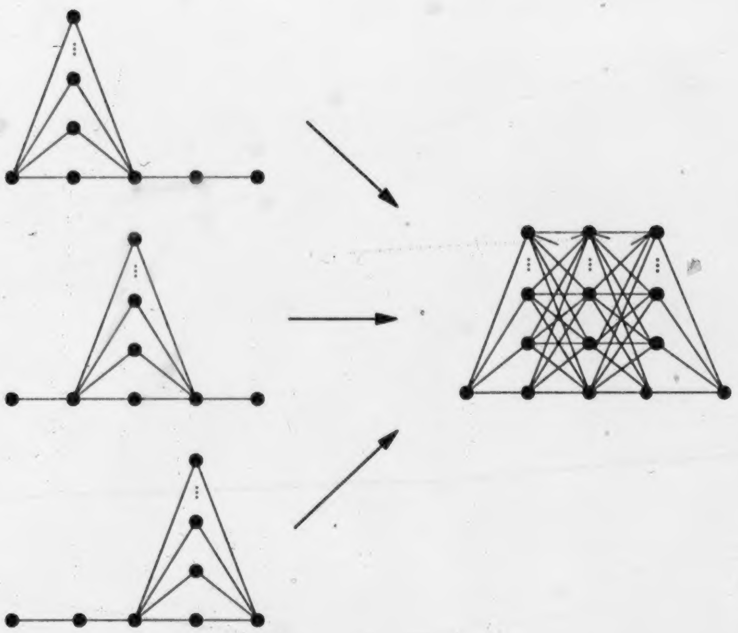


Figure 3.3.7: Synthesis of a subtrellis for the two-level squaring construction.

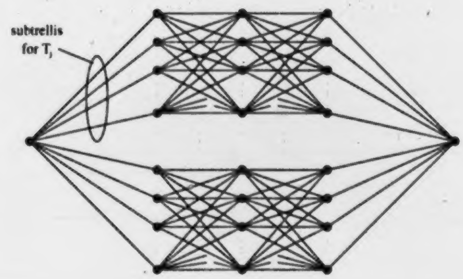

subtrellis
for T_j

Figure 3.3.8: A complete trellis based on the two-level squaring construction.

RM(1,1)	-	RM(2,2)	-	RM(3,3)	-	RM(4,4)	-	RM(5,5)	-	RM(6,6)
| 2		| 2		| 2		| 2		| 2		| 2
RM(0,1)	-	RM(1,2)	-	RM(2,3)	-	RM(3,4)	-	RM(4,5)	-	RM(5,6)
| 2		| 4		| 8		| 16		| 32		| 64
RM(-1,1)	-	RM(0,2)	-	RM(1,3)	-	RM(2,4)	-	RM(3,5)	-	RM(4,6)
		| 2		| 8		| 64		| 1024		| 32768
		RM(-1,2)	-	RM(0,3)	-	RM(1,4)	-	RM(2,5)	-	RM(3,6)
				| 2		| 16		| 1024		| 1048576
				RM(-1,3)	-	RM(0,4)	-	RM(1,5)	-	RM(2,6)
						| 2		| 32		| 32768
						RM(-1,4)	-	RM(0,5)	-	RM(1,6)
								| 2		| 64
								RM(-1,5)	-	RM(0,6)
										| 2
										RM(-1,6)

Table 3.1: RM (ρ,ν) codes and their partition degrees.

structed using the two-level squaring operation

$$\mathrm{RM}\,(2,4) = |\mathrm{RM}\,(2,2)\,/\mathrm{RM}\,(1,2)\,/\mathrm{RM}\,(0,2)|^4. \tag{3.3.5}$$

An RM $(0,2)$ code consists of two length-four vectors, e.g. $\langle 0000\rangle$ and $\langle 1111\rangle$. The matrix C therefore consists of a single row, $\langle 1111\rangle$.

As indicated in Table 3.1, there are four coset selectors for the partition RM $(1,2)$ /RM $(0,2)$. These selectors can be chosen to be $a = \langle 1111\rangle$, $b = \langle 0110\rangle$, $c = \langle 1010\rangle$, and $d = \langle 1100\rangle$. Only two of these are linearly independent. The matrix Q therefore has two rows, which we choose to be b and c.

The partition RM $(2,2)$ /RM $(1,2)$ has degree two, and thus has only one non-zero coset selector. The only unrepresented vector left is $e = \langle 0001\rangle$. The matrix Q_2 consists of a single row, e.

The subtrellis is now constructed based on the generator matrix

$$G_H = \begin{bmatrix} b & b & & \\ c & c & & \\ & b & b & \\ & c & c & \\ & & b & b \\ & & c & c \end{bmatrix}. \tag{3.3.6}$$

The resulting coset subtrellis for G_H is shown in Figure 3.3.9. Each branch in Figure 3.3.9 has two labels, l and $l + \langle 1111\rangle$.

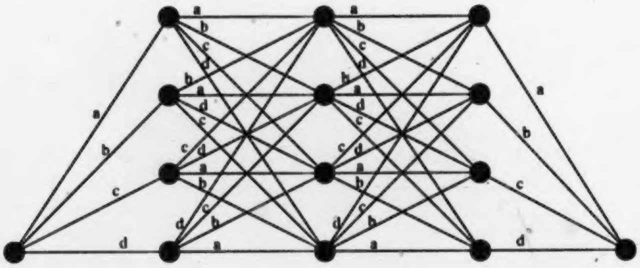


Figure 3.3.9: Coset subtrellis for (16,11) Hamming code. The complete trellis includes a second, identical subtrellis connected as in Figure 3.3.8.

A second coset subtrellis is generated by adding $e = \langle 0001 \rangle$ to every branch label in the subtrellis:

$$a = \left\{ \begin{array}{c} 1111 \\ 0000 \end{array} \right\} \longrightarrow ea = e + a = \left\{ \begin{array}{c} 1110 \\ 0001 \end{array} \right\}. \tag{3.3.7}$$

The two coset subtrellises are merged at S_0, which completes the trellis.

To use this trellis in a sum-product decoder, we require a complete factor graph which shows the relationships among information bits, parity bits, and the trellis labels. Let $z_1, \ldots, z_4$ be a contiguous sequence of four channel bits. The constraint between $\langle z_1, z_2, z_3, z_4 \rangle$ and l is depicted in the trellis-style graph of Figure 3.3.10.

With all constraints specified, we have sufficient information to construct a sum-product decoder. The completed factor graph for this code is shown in Figure 3.3.11, in which the boxes labeled '1', '2' and '3' are sections of the label constraint trellis (Figure 3.3.10), and the solid boxes represent the two center sections of the constructed trellis.

The double-circles indicate hidden variable nodes. These only serve as labels, and do not have any computational significance. The order of the variable nodes $z_1, \ldots, z_{16}$ has also been permuted in some places. This is done to ensure a systematic code. The necessary permutation is revealed by a construction of the code's generator matrix.

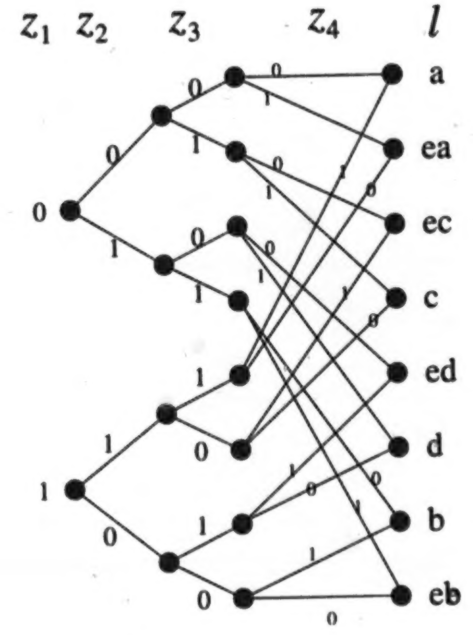


Figure 3.3.10: A trellis-style graph indicating the constraints among channel bits z_i and branch label l.

The generator matrix for the constructed (16,11) Hamming code is

$$G_{16} = \begin{bmatrix} a & & & \\ & a & & \\ & & a & \\ & & & a \\ e & e & e & e \\ b & b & & \\ c & c & & \\ & b & b & \\ & c & c & \\ & & b & b \\ & & c & c \end{bmatrix} = \begin{bmatrix} 1111 & & & \\ & 1111 & & \\ & & 1111 & \\ & & & 1111 \\ 0001 & 0001 & 0001 & 0001 \\ 0110 & 0110 & & \\ 0011 & 0011 & & \\ & 0110 & 0110 & \\ & 0011 & 0011 & \\ & & 0110 & 0110 \\ & & 0011 & 0011 \end{bmatrix}.$$

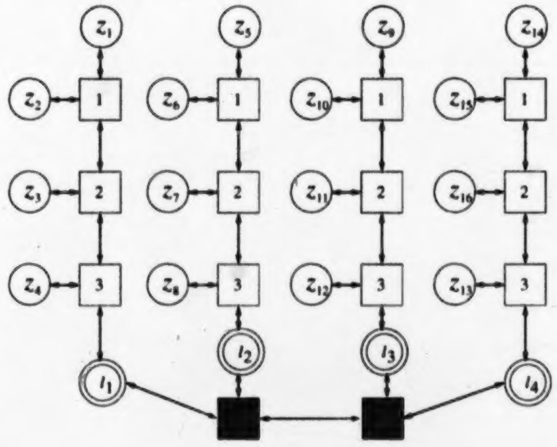


Figure 3.3.11: Complete factor graph for the (16,11) Hamming code.

After a sequence of linear row operations, G_{16} becomes

$$G_{16} = \begin{bmatrix} 1000 & 0001 & 0001 & 0111 \\ 0100 & 0001 & 0001 & 0100 \\ 0010 & 0001 & 0001 & 0001 \\ 0001 & 0001 & 0001 & 0010 \\ & 1001 & & 0110 \\ & 0101 & & 0101 \\ & 0011 & & 0011 \\ & & 1001 & 0110 \\ & & 0101 & 0101 \\ & & 0011 & 0011 \\ & & & 1111 \end{bmatrix}. \qquad (3.3.8)$$

G_{16} is nearly systematic, except for parity-check columns in positions 8 and 12. The matrix can be made systematic by a column transposition:

$$(16,11) \text{ systematic transposition} = (1,2,3,4,5,6,7,9,10,11,13,8,12,14,15,16). \qquad (3.3.9)$$

The construction is now complete. The trellis constraints are implemented as sum-product nodes using the transition-matrix method, as discussed in Section 3.2.2.

3.3.5 *Tailbiting squaring construction.

The two-level squaring construction can be also used to construct tailbiting trellises for Hamming and other RM codes. The resulting tailbiting trellises have a smaller

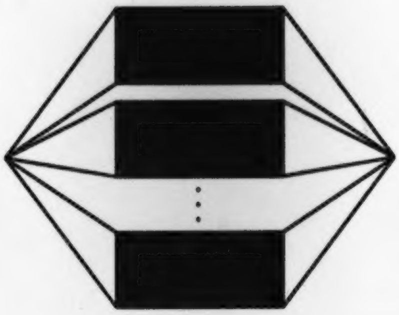


Figure 3.3.12: Shape of conventional trellises constructed with the two-level squaring procedure.

maximum state complexity than the conventional trellises produced by the same method. For large codes, the state-complexity can be reduced by 50%.

Recall from Section 3.3.3 that the trellis consists of disjoint coset subtrellises. A coset subtrellis is completely determined by the matrix Q, as in (3.3.3). Let the rows of Q be $q_1, \ldots, q_N$. Then the generator matrix which corresponds to the coset subtrellis is

$$G_S = \left[\begin{array}{cccc} \underline{q_1} & q_1 & & \\ \vdots & \vdots & & \\ \underline{q_N} & q_N & & \\ & \underline{q_1} & q_1 & \\ & \vdots & \vdots & \\ & \underline{q_N} & q_N & \\ & & \underline{q_1} & q_1 \\ & & \vdots & \vdots \\ & & \underline{q_N} & q_N \end{array}\right].$$

The state-complexity profile for G_S is $(N, N, N, 0)$, and the number of states at the corresponding time-indices are $2^N, 2^N, 2^N, 1$. The shape of a trellis constructed using the conventional procedure is shown in Figure 3.3.12.

The coset subtrellis can be made tailbiting through a few row operations on G_S.

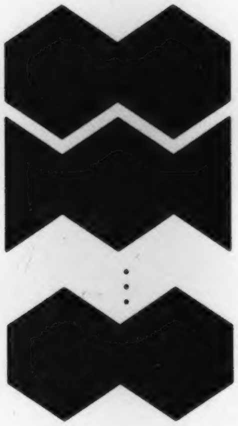


Figure 3.3.13: Shape of tailbiting trellises constructed with the two-level squaring procedure.

We split G_S horizontally at its midpoint, and separate the center Q matrices:

$$G_T = \left[\begin{array}{ccccc}
 & \underline{q_1} & q_1 & & \\
 & \vdots & \vdots & & \\
 & \underline{q_N} & q_N & & \\
 & & \underline{q_1} & q_1 & \\
 & & \vdots & \vdots & \\
 & & \underline{q_{\frac{N}{2}}} & q_{\frac{N}{2}} & \\
\cline{2-4}
q_{\frac{N}{2}+1} & & & & \underline{q_{\frac{N}{2}+1}} \\
\vdots & & & & \vdots \\
q_N & & & & \underline{q_N} \\
 & & \underline{q_1} & & q_1 \\
 & & \vdots & & \vdots \\
 & & \underline{q_N} & & q_N
\end{array}\right]. \qquad (3.3.10)$$

A trellis constructed from (3.3.10) has state-complexity profile $\left(N,\frac{N}{2},N,\frac{N}{2}\right)$. The matrix G_T can also be easily manipulated to produce state-complexity profile $\left(\frac{N}{2},N,\frac{N}{2},N\right)$. By alternating these profiles between the disjoint coset subtrellises, a more efficient state-complexity is obtained, as shown in Figure 3.3.13.

Suppose there are M coset subtrellises. Then a conventional trellis constructed as in Section 3.3.3 has a maximum state-depth of $M \cdot 2^N$. A tailbiting trellis constructed using our procedure has a maximum state-depth of $\frac{M}{2} \cdot \left(2^N + 2^{\frac{N}{2}}\right)$. The

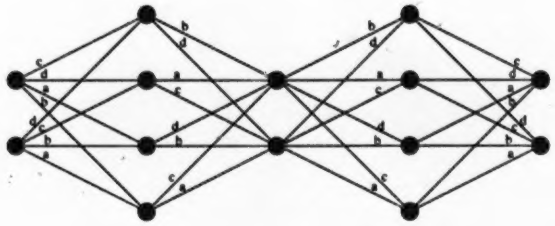


Figure 3.3.14: Tailbiting form of Figure 3.3.9.

complexity ratio between the two is

$$\frac{2^{N-1}+2^{\frac{N}{2}-1}}{2^N}=\frac{1}{2}+\frac{\sqrt{2^{-N}}}{2}\rightarrow\frac{1}{2}.$$

The tailbiting squaring construction therefore reduces maximum state-complexity by up to 50%.

Trellises constructed with this method are completely uniform, in that they are repetitions of the same trellis sections, and they have the same state-complexity at each time-index.

Example 3.3.4. We now apply the squaring method to construct a tailbiting trellis for a (16,11) Hamming code. The generator matrix corresponding to a coset subtrellis, derived from (3.3.6), is

$$G_H=\begin{bmatrix} b & b & & \\ c & c & & \\ & b & b & \\ c & & & c \\ & & b & b \\ & & c & c \end{bmatrix}. \tag{3.3.11}$$

The tailbiting coset subtrellis constructed from G_H is shown in Figure 3.3.14. The complete tailbiting trellis for the (16,11) code is shown in Figure 3.3.15.

□

3.4 *True-MAP decoding on tailbiting trellises.

For a tailbiting trellis, the *exact* trellis state probabilities can be obtained through a calculation which excludes pseudocodewords. In general, to calculate the exact

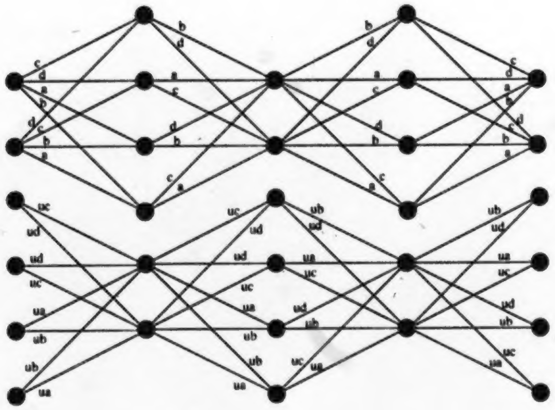


Figure 3.3.15: Complete tailbiting trellis for the (16,11) Hamming code.

state probabilities for one time-index, we must compute the product of all transition matrices in the trellis. This calculation must be done separately for each time-index, resulting in a very complex set of operations.

In certain special cases, however, the structure of the transition matrices allows the set of products to be computed more efficiently. In some cases, the exact solution actually has lower complexity than the BCJR algorithm. This is the case for some very small codes. A more efficient decoder for small codes may have application when they form the basis of larger codes, such as "braided" Turbo codes.

A general exact MAP algorithm for tailbiting trellises is as follows. Recall from Section 3.2.2 that $A_0 = \prod_{i=1}^{L} \Gamma_i$. The matrix A_0 can be interpreted as the transition matrix of the trellis, with all sections merged. It thus represents a single trellis section $\mathcal{T}_{A_0} = (\mathcal{S}_0, \mathcal{L}_C, \mathcal{S}_L)$, where $\mathcal{L}_C$ is the set of all connected sequences in the trellis. But $\mathcal{S}_L = \mathcal{S}_0$, so $\mathcal{T}_{A_0} = (\mathcal{S}_0, \mathcal{L}_C, \mathcal{S}_0)$.

The elements a_{ij} of A therefore represent the conditional probability of traversing a path from $\mathbf{s}_0 = s_0^{(i)}$ to $\mathbf{s}_0 = s_0^{(j)}$. The only genuine paths through the trellis are those which connect $\mathbf{s}_0 = s_0^{(i)}$ to $\mathbf{s}_0 = s_0^{(i)}$, corresponding to the diagonal elements of A_0. All non-diagonal elements correspond to pseudocodewords.

To obtain the exact state probabilities, it therefore suffices to discard all but the diagonal elements of A_0. Let diag(A_0) be a row vector whose i^{th} member is equal to a_{ii}. Then

$$\underline{\sigma}_0 = \text{diag}(A_0) \tag{3.4.1}$$

The remaining state probabilities $\underline{\sigma}_1, \ldots, \underline{\sigma}_{L-1}$ are found by taking cyclic shifts of the Γ_i products. Thus

$$\begin{aligned} A_i &= \prod_{j=1}^{L} \Gamma_{(j+i) \bmod L} && (3.4.2) \\ \underline{\sigma}_i &= \text{diag}\,(A_i) && (3.4.3) \end{aligned}$$

3.4.1 *Complexity of the exact MAP algorithm.

In general, computing the permuted matrix products specified in (3.4.2) is very complex. A matrix product $\Gamma_i\Gamma_{i+1}$ requires m^2 multiplications, where m is the number of distinct branch labels in a trellis section. The BCJR algorithm requires one multiplication per branch in a trellis section.

Example 3.4.1. In the squared tailbiting trellis construction described in Section 3.3.5, each trellis section has 2^N distinct branch labels. There are consequently 2^{2N} multiplications required for a transition matrix product. Each section has $2^{\frac{3N}{2}}$ branches, which is the number of multiplications needed for one stage of the BCJR algorithm.

□

As the trellis's state-depth and length increase, the complexity of the exact MAP algorithm grows more rapidly than that of BCJR. For shorter, simpler trellises, however, less complex solutions are obtainable with the exact MAP approach. We illustrate this with an example optimization for the (8,4) Hamming code.

3.4.2 *Exact MAP for tailbiting (8,4) Hamming trellis.

A tailbiting trellis for an (8,4) Hamming code is shown in Figure 3.4.1. In this figure, line-styles (e.g. solid, dotted, dash-dot, dashed) indicate the channel symbol. Two branches are connected to the right of each state. The upper branch corresponds to an information symbol of '0' and the lower branch to an information

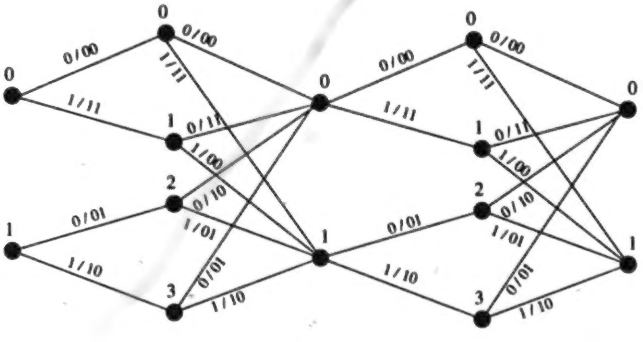


Figure 3.4.1: Tailbiting trellis for (8,4) Hamming code.

symbol of '1'. These relationships are indicated explicitly in the left-most section, and are implicit in the others.

The trellis sections T_0 through T_3 are indicated in Figure 3.4.1, along with the corresponding transition matrices Γ_0 through Γ_3, and the state-variables S_0 through S_3. The tailbiting termination is indicated by the label S_0 on both the left-most and right-most state variables.

In this trellis, MAP decisions can be taken directly from the state-probabilities $\underline{\sigma}_i$. The MAP solution for $\underline{\sigma}_i$ is therefore equivalent to MAP decoding. Applying (3.4.2) and (3.4.3), the MAP solutions for each state are found by

$$\begin{aligned} \underline{\sigma}_0 &= \operatorname{diag}(\Gamma_0\Gamma_1\Gamma_2\Gamma_3) \qquad & \underline{\sigma}_2 &= \operatorname{diag}(\Gamma_2\Gamma_3\Gamma_0\Gamma_1) \\ \underline{\sigma}_1 &= \operatorname{diag}(\Gamma_1\Gamma_2\Gamma_3\Gamma_0) \qquad & \underline{\sigma}_3 &= \operatorname{diag}(\Gamma_3\Gamma_0\Gamma_1\Gamma_2) \end{aligned}$$

These products are simple enough that they may be carried out by hand. It is easily verified that the products $(\Gamma_1\Gamma_2)$ and $(\Gamma_3\Gamma_0)$ each have sixteen distinct products. In computing $\operatorname{diag}[(\Gamma_1\Gamma_2)(\Gamma_3\Gamma_0)]$, we find that an additional sixteen distinct products are produced. Each of these products represents a codeword. There are a total of sixteen codewords in the code. Once we have solved for $\underline{\sigma}_1$, therefore, no further multiplications are required. The remaining solutions require only addition over these products.

We will now begin to count operations. We use the notation $\{x\odot, y\oplus\}$ to indicate x multiplications and y additions. In obtaining $\underline{\sigma}_1$, we require $\{48\odot, 12\oplus\}$. It is left as an exercise to verify that $\underline{\sigma}_3$ requires $\{12\oplus\}$ and $\underline{\sigma}_0$ requires $\{12\oplus\}$. By judicious arrangement of additions, $\underline{\sigma}_2$ can then be obtained with only $\{2\oplus\}$

additional operations. The total required operations is $\{48\odot, 38\oplus\}$.

For comparison, the BCJR algorithm requires $\{24\odot, 12\oplus\}$ operations for a forward or backward pass around the trellis. To approximate state probabilities, BCJR requires one additional multiplication per state, which adds $\{12\odot\}$. Because BCJR is approximate on tailbiting trellises, it is usually necessary to extend the propagation more than once around the trellis. If we use the optimistic estimate of 1.5 passes around the trellis, then BCJR requires a total of $\{84\odot, 36\oplus\}$. The exact MAP solution for this trellis therefore requires 43% fewer multiplications than the BCJR algorithm.

No text

Chapter 4

LDPC Codes

Low-Density Parity Check (LDPC) codes were first introduced by Gallager in 1962 [33]. It was not until very recently, mainly through the work of MacKay [56], that the full potential of these codes was discovered. Using probability propagation along a code's graph, suitably chosen LDPC codes can rival the performance of Turbo codes with similar complexity [69, 70].

An LDPC code is a large linear block code with a sparse matrix of parity checks. The design methods used for selecting LDPC codes and their decoders may be thought of as a prescription for selecting large random block codes, not unlike the random codes suggested in the derivation of Shannon's coding theorem [78]. It has also been shown that certain ensembles of LDPC codes approach the Shannon limit exponentially fast in code length for some channels. Ensembles are also known which converge exponentially in code length to limits within 0.1dB of the Shannon limit on the additive white Gaussian noise (AWGN) channel with binary signaling. Codes are known which, on the AWGN channel, come within 0.04 dB of the Shannon limit at a code length of 10^7, within 0.1dB at 10^6, and within 1dB at 10^4[69, 20].

4.1 Regular and Irregular Ensembles

4.1.1 Regular LDPC codes

The earliest LDPC codes were defined by Gallager in terms of their parity-check matrices. A (d_v, d_c)-Gallager code of length n, as defined in [70] has an $n \times m$

parity check matrix with random entries such that each row has exactly d_c 1's and each column has exactly d_v 1's. The number of rows m is given by $m = n\frac{d_v}{d_c}$. The parameters d_v and d_c are chosen to be small, so that the resulting parity-check matrix is sparse (hence the term "low-density"). In the corresponding graph, all variable nodes have degree d_v and all check nodes have degree d_c. Such a code is also referred to as a **$(\mathbf{d_v}, \mathbf{d_c})$-regular LDPC code**.

LDPC codes are conventionally discussed in terms of their non-normal Tanner graphs. As explained in Section 2.3.2, the *normal form* of a Tanner graph introduces equality constraints at the variable nodes. In keeping with the LDPC literature, in this Chapter we often refer to the sum-product "processing" at variable nodes. This processing is actually performed by an implicit equality node.

Note that the pair (d_v, d_c), together with the code length n, specifies an *ensemble* of codes, rather than any particular code. This ensemble is denoted by $C^n(d_v, d_c)$. Once we know the degrees of the nodes, we are still free to choose which particular connections are made in the graph. Some results indicate that the best way to choose these connections is at random[69]. It is perhaps more appropriate to reword this result as follows: a formal construction procedure is *unlikely* to produce a better code than those which could be selected at random. This result is more than conjecture, but many researchers still prefer construction procedures which reduce the complexity of the decoder or encoder, and which reduce the occurrence of short cycles. Such methods may not produce *better codes*, but from many perspectives they may result in *better decoders* in terms of designability and efficient resource utilization.

A *socket* refers to a point on a node to which an edge may be attached. For example, we say that a variable node has d_v sockets, meaning d_v edges may be attached to that node. There will be a total of nd_v sockets on the left (variable) side of any code graph G in $C^n(d_v, d_c)$. The right (check) side of G must also have nd_v sockets, as this is the number of edges in the graph. A particular pattern of edge connections can therefore be described as a permutation π from left sockets to right sockets. An individual edge is specified by the pair $(i, \pi(i))$, which indicates that the i^{th} left socket is connected to the $\pi(i)^{th}$ right socket.

Selecting a random code from the ensemble $C^n(d_v, d_c)$ therefore amounts to randomly selecting a permutation on nd_v elements. Many permutations will result in a graph which contains *parallel edges*, i.e. in which more than one edge join the same variable and parity check nodes. Note that, in the parity check matrix, an even number of parallel edges will cancel. If they are deleted from the graph, then the degrees of some nodes will be changed and the code will cease to be a regular LDPC code. If not deleted, their presence will ruin the performance of message-passing decoding algorithms. We must therefore make the restriction that permutations leading to parallel edges are disallowed.

The *design rate* of a regular LDPC code is defined as

$$r = \frac{n-m}{n} = 1 - \frac{d_v}{d_c}. \tag{4.1.1}$$

The design rate may differ from the actual code rate in that some check nodes may be redundant.

4.1.2 Irregular LDPC Codes

Under probability propagation decoding, the performance of regular LDPC codes is disappointing. A Turbo code of similar size will outperform a regular LDPC code by about 0.5dB at BER=10^{-6}. It has been found that by allowing node degrees to vary, LDPC codes' performance using sum-product decoding may exceed that of Turbo codes. LDPC codes with non-constant node degrees are called irregular codes [69].

An irregular code cannot be defined in terms of the degree parameters d_v and d_c. We must instead use *degree distributions* to describe the variety of node degrees in the graph. A degree distribution $\gamma(x)$ is a polynomial:

$$\gamma(x) = \sum_i \gamma_i x^{i-1} \tag{4.1.2}$$

such that $\gamma(1) = 1$. The coefficients γ_i denote the fraction of *edges* in the graph which are connected to a node $n \in \mathcal{N}$ of degree i, where $\mathcal{N}$ is a subset of the graph's nodes. Usually $\mathcal{N}$ is the set of all check nodes, or the set of all equality/variable nodes. The degree distribution thus describes the pattern of degrees among nodes in one *layer* of the code's Tanner graph.

We also use the notation $\int\gamma$ to denote the average inverse degree, which is defined as

$$\int\gamma \equiv \int_0^1 \gamma(x)dx = \sum_i \frac{\gamma_i}{i}. \tag{4.1.3}$$

To better understand (4.1.3), recall that γ_i is the fraction of *edges* with degree i. The fraction of *nodes* with degree i, denoted by $\widetilde{\gamma}_i$, is proportional to $\frac{\gamma_i}{i}$. We require $\sum_i \widetilde{\gamma}_i = 1$. We use (4.1.3) as the proportionality constant, arriving at

$$\widetilde{\gamma}_i = \frac{\gamma_i}{i\int\gamma}. \tag{4.1.4}$$

The code length n and two degree distributions – λ and ρ for the variable and check nodes, respectively – are sufficient to define an ensemble $C^n(\lambda,\rho)$ of irregular LDPC codes. A graph G from this ensemble will have n variable nodes. The number of check nodes, m, is given by:

$$m = n\frac{\int\rho}{\int\lambda}. \tag{4.1.5}$$

The number of degree-i variable nodes in G is

$$\widetilde{\lambda}_i = \frac{\lambda_i}{i\int\lambda}. \tag{4.1.6}$$

Similarly, the number of degree-i check nodes in G is

$$\widetilde{\rho}_i = \frac{\rho_i}{i\int\rho}. \tag{4.1.7}$$

And the design rate of the code represented by G is

$$r = \frac{n-m}{n}. \tag{4.1.8}$$

Again, the design rate may differ from the actual rate. We can once again enumerate the left and right sockets of the irregular code graph. Selection of a code from the ensemble is a selection of a permutation on N_E elements, where N_E is the number of edges in the graph, given by

$$N_E = \frac{n}{\int \lambda}. \tag{4.1.9}$$

We rule out parallel edges in irregular codes as well.

4.2 Density Evolution and Code Selection

The notation defined so far allows us to define ensembles of LDPC codes, but how do we know which parameters to choose and which codes to pick for good performance? The authors of [70] use *density evolution* to compare the qualities of different ensembles of regular and irregular LDPC codes. They consider these ensembles on channels which can be ordered by a single *channel parameter s* (such as N_0 for the AWGN channel, or the crossover probability ε for the BSC). It is convenient to define this parameter in such a way that the channel improves for decreasing s.

Each degree distribution pair (λ, ρ), on a specific channel $\mathcal{C}(s)$ ordered by parameter s, has an associated *threshold* s^*. The threshold is analogous to capacity: the error-probability for transmission on $\mathcal{C}(s)$ for a randomly chosen (λ, ρ)-code can be made arbitrarily small if and only if $s < s^*$. One degree-distribution pair is better than another for a specified channel if its threshold is closer to the channel's capacity limit.

In designing good codes, we choose the ensemble with the best threshold, from which we select a code with the largest feasible length. It also turns out that almost all codes in the ensemble perform equally well. Code design may therefore consist of randomly sampling a few codes from the ensemble and selecting the best among those.

The density evolution method is outlined as follows:

1. The channel is specified in terms of a single parameter s, and a decoding algorithm is specified.
2. Assume the limit of infinitely long codes, in which the corresponding Tanner graph is cycle-free.

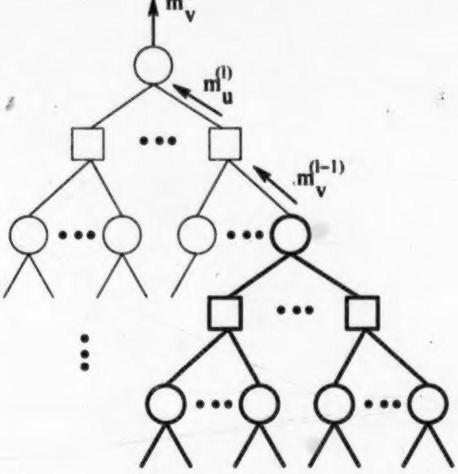


Figure 4.2.1: Tree representation of an infinitely large LDPC code.

3. Represent the decoder as a tree T_l. Note that the subtree T_{l-1} is indistinguishable from T_l, as illustrated in Figure 4.2.1. A subtree T_{l-1} is indicated in bold in Figure 4.2.1

4. Based on the decoding algorithm, find a transformation $\mathcal{F}$ which relates the probability density function of a message at iteration l to the density at iteration $l-1$. Thus if X is a message and $p[X]$ is the probability density function of X, then $p\left[m_v^{(l)}\right] = \mathcal{F}\left\{p\left[m_v^{(l-1)}\right]\right\}$. The initial density $m_v^{(0)}$ is determined by the channel.

5. Based on the relation $\mathcal{F}$, determine the set of parameters s for which the density of *incorrect messages* converges to zero, i.e. the set of s such that the *error probability* converges to zero. The boundary of this set, s^*, is the ensemble's threshold.

4.3 The AWGN channel and the Gaussian Approximation.

The exact solution for density evolution is quite involved, but a close approximation can be found with little effort if all messages in the decoder are assumed to have a Gaussian density. This is in general not the case, but the Gaussian approximation greatly reduces the complexity and gives results which are close to the exact solution.

When binary transmission is used on an AWGN channel, the probability density of *log-likelihood ratio* messages at the channel's output is

$$f_0(y) = \sqrt{\frac{N_0}{4\pi}} e^{-\frac{N_0}{4}\left(y - \frac{1}{N_0}\right)}. \tag{4.3.1}$$

For a density $f_Y(y)$ of this form for a log-likelihood message Y, we can derive the *symmetry conditions* [70]

$$\begin{aligned} f_Y(-y) &= f_Y(y)e^{-y}, & (4.3.2) \\ m_Y &= \frac{\sigma_Y^2}{2} & (4.3.3) \end{aligned}$$

where m_Y and σ_Y^2 are the mean and variance of Y, respectively.

The condition (4.3.3) indicates that the density f_Y is completely determined by the mean m_Y. The Gaussian assumption thus allows us to consider the evolution of a single decoder statistic.

This approach has another convenient property: in the log-likelihood domain, there is a one-to-one relationship between the mean equality node output and the error probability. If we assume that the all-zero codeword has been transmitted, then all *correct* messages should be *negative*. The error probability is therefore simply

$$P_e = Q\left(-\sqrt{\frac{m_v}{2}}\right) \tag{4.3.4}$$

where $Q(x)$ is the well-known Gaussian integral from x to infinity, and m_v is the mean variable node message.

We can also describe this procedure for messages in the probability domain, in which a node's output is a pair of probabilities p_0 and p_1 instead of a log-likelihood ratio. Because the all-zero codeword is transmitted, a correct message should satisfy $p_0 > p_1$. The error probability is therefore the average p_1 message at the variable node's output.

In terms of computation, it is helpful to use both LLR and probability messages. There is a one-to-one correspondence between the two, so it is fair to switch between them if doing so will improve the computation. Care must of course be taken to avoid biased averages. If we generate an LLR from the average probability message, the result is not necessarily the average LLR message. Such conversion must take place before computing any averages.

The key problem of density evolution is to determine whether the error probability converges to zero. Let $r^{(l)}$ be the error probability at the output of a variable node after l iterations, and recall that s is the channel's noise parameter, which for a Gaussian channel is

$$s \equiv \frac{1}{N_0}. \tag{4.3.5}$$

In keeping with the notation used in [21], we define the function $h(s,r)$ as the error probability after one iteration, given r as the initial error probability. Thus

$$r^{(l+1)} = h\left(s, r^{(l)}\right). \tag{4.3.6}$$

This recursive relationship induces a sequence of error probabilities, $\left\langle r^{(0)} = s,\, r^{(1)},\, r^{(2)},\, \ldots \right\rangle$. For error-free decoding, this sequence should converge to zero.

It is well known from the theory of one-dimensional iterated maps that a recursive sequence has a fixed point greater than zero if and only if there is a point $r_p > 0$ for which $r_p = h(s, r_p)$. This concept is visualized in Figure 4.3.1, which shows various curves for $h(s,r)$ along with the *unit line* $h(r) = r$. If $h(s,r)$ does not intersect the unit line, then the error probability converges to zero. If $h(s,r)$ intersects the unit line, as illustrated by the dashed curve, then there is a fixed point greater than zero.

For large s (and therefore small σ), the error probability converges to zero. For small s (large σ), it does not. The boundary between convergence and non-

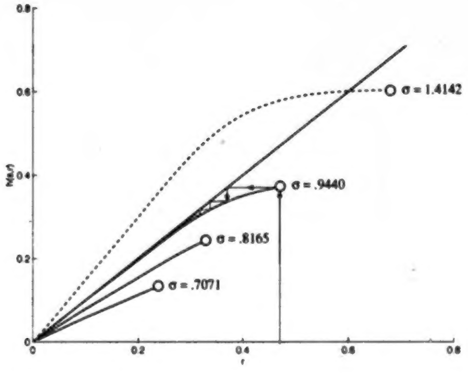


Figure 4.3.1: Iterative behavior of $h(s, r)$.

convergence is the threshold.

4.3.1 Computing the threshold.

In this section we review the procedure of [21], which allows density evolution to be carried out with minimal computation. The analytical results in this section assume that "pure" sum-product decoding is used. This analysis therefore cannot account for practical issues such as finite-precision arithmetic or approximate versions of the algorithm.

Thresholds calculated using this procedure are usually very close to the exact thresholds for LDPC code ensembles. Although the true message probability densities are not *exactly* Gaussian, very low error results from assuming that they are [21]. We may therefore reasonably assume that the results provide a good estimate of the bound on achievable performance for an LDPC ensemble.

We first examine density evolution in variable nodes. For variable nodes under probability propagation in the log domain, the output message is simply the sum of messages incoming on other edges. We assume the messages are independent and identically distributed (i.i.d.), so that

$$m_v^{(l)} = m_v^{(0)} + (d_v - 1)m_u^{(l-1)} \tag{4.3.7}$$

where $m_v^{(l)}$ is the mean LLR output from the variable node at iteration l, and $m_u^{(l-1)}$ is the mean LLR output from a check node at iteration $(l-1)$, and $m_v^{(0)} = s = \frac{1}{N_0}$.

The density evolution through a check node is more complicated. If V_i is an incoming LLR message to the check node, $i \in \{1, ..., d_c - 1\}$, and U is the outgoing LLR message, the update rule for a check node is

$$\tanh\left(\frac{U}{2}\right) = \prod_{i=1}^{d_c-1} \tanh\left(\frac{V_i}{2}\right). \tag{4.3.8}$$

To find the evolution of the mean LLR, we assume the messages are i.i.d and take the expectation of both sides:

$$E\left[\tanh\left(\frac{U}{2}\right)\right] = E\left[\tanh\left(\frac{V_i}{2}\right)\right]^{d_c-1}. \tag{4.3.9}$$

To simplify the analysis, we define a function ϕ as

$$\phi(m_u) = 1 - E\left[\tanh\left(\frac{U}{2}\right)\right] \tag{4.3.10}$$

$$= \begin{cases} 1 - \frac{1}{\sqrt{4\pi m_u}} \int_{(R)} \tanh\left(\frac{u}{2}\right) \exp\left[-\frac{1}{4m_u}(u-m_u)^2\right] du & \text{if } m_u > 0 \\ 1 & \text{otherwise} \end{cases} \tag{4.3.11}$$

The ϕ function looks complicated and surprising, but it has a simple explanation. In Section 4.3 it was mentioned that the LLR of the mean probabilities is not equal to the mean LLR. Some more complicated relation is required to transform between them. This relation is expressed by the ϕ function:

$$\phi(E[\text{LLR}]) = E[p_1]. \tag{4.3.12}$$

By using the one-to-one functions ϕ and ϕ^{-1}, we may easily switch between mean-LLR and mean-probability messages. Unfortunately, ϕ or ϕ^{-1} must be approximated, either numerically or by some very inexact formulas. The author of this thesis uses Monte Carlo estimation to compute ϕ.

By converting from LLRs to probabilities, and switching between p_0 and p_1 whenever it is convenient, we find that the mean of the check node's output message is

$$m_u^{(l-1)} = \phi^{-1}\left\{1 - \left[1 - \phi\left(m_v^{(l-1)}\right)\right]^{d_c-1}\right\}. \tag{4.3.13}$$

The results (4.3.7) and (4.3.13) are sufficient to compute thresholds for regular ensembles. We treat node degrees as random variables. The message densities are now assumed to be Gaussian mixtures, with (4.3.7) and (4.3.13) as partial solutions. Combining these partial solutions to produce the general density evolution for irregular ensembles, we arrive at

$$m_u^{(l-1)} = \sum_{i=1}^{d_c} \rho_i \cdot \phi^{-1}\left\{1 - \left[1 - \phi\left(m_v^{(l-1)}\right)\right]^{i-1}\right\} \tag{4.3.14}$$

$$m_v^{(l)} = s + \sum_{j=1}^{d_v} \lambda_j \cdot (j-1) \cdot m_u^{(l-1)}. \tag{4.3.15}$$

4.3.2 Threshold Determination

Equations 4.3.14 and 4.3.15 describe density evolution in the LLR domain. It is much easier to work with evolution in the probability domain, as in (4.3.6). To convert to the probability domain, we simply use the ϕ function:

$$h_j(s,r) = \phi\left[s + (j-1)\sum_{i=1}^{d_r} \rho_i \cdot \phi^{-1}\left(1 - (1-r)^{i-1}\right)\right] \tag{4.3.16}$$

$$h(s,r) = \sum_{j=1}^{d_v} \lambda_j \cdot h_j(s,r). \tag{4.3.17}$$

Note that for a regular ensemble, there is only one h_j.

The threshold is explicitly defined as

$$s^* = \inf\left\{s \in \mathbb{R}^+ : h(s,r) - r < 0, \forall r \in (0, \phi(s))\right\}. \tag{4.3.18}$$

Using (4.3.18), the threshold can be found numerically using a simple search algorithm. The search is made easier by examining the curve $h(s,r) - r$, which always has the shape shown in Figure 4.3.2. The threshold is the value of s for which $h(s,r) - r$ barely touches zero at its maximum.

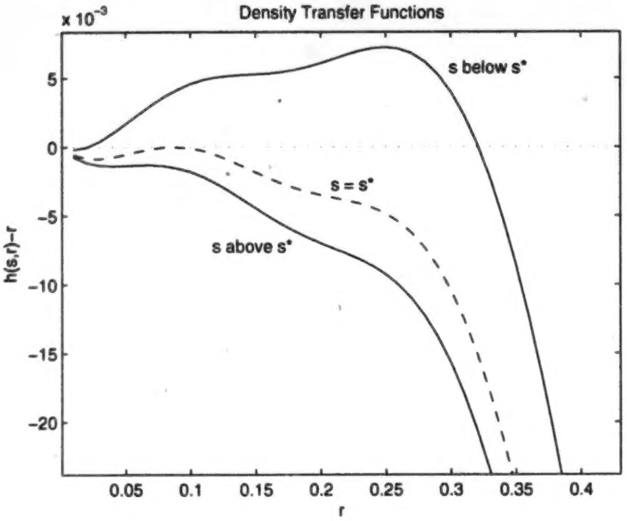


Figure 4.3.2: A detailed view of $h(s,r) - r$ when s is just above, just below, and equal to the threshold. These curves are for the irregular LDPC ensemble defined in the first (leftmost) column of Table 4.2.

Approximations

Exact computation of ϕ and ϕ^{-1} can be computationally expensive. The computation speed can be greatly improved using a few approximations. For small x, perhaps $x < 10$, a good approximation for ϕ is

$$\phi(x) \approx e^{\alpha x^{\gamma} + \beta} \tag{4.3.19}$$

where $\alpha = -.4527$, $\beta = .0218$, $\gamma = .86$. For larger x, the following upper and lower bounds become tight, so that their average can be used as a good approximation to ϕ:

$$\sqrt{\frac{\pi}{x}} e^{-\frac{x}{4}} \left(1 - \frac{3}{x}\right) < \phi(x) < \sqrt{\frac{\pi}{x}} e^{-\frac{x}{4}} \left(1 - \frac{1}{7x}\right). \tag{4.3.20}$$

Tables 4.1 and 4.2 present thresholds for a variety of rate-$\frac{1}{2}$ irregular LDPC ensembles. Exact thresholds from [69] are presented alongside thresholds computed using the Gaussian message approximation and the ϕ approximations (4.3.19) and (4.3.20). The maximum observed error in these estimates is 0.19dB.

Table 4.1: Exact (SNR^*) [69] and Gaussian-approximation (SNR^*_{GA}) thresholds for various rate-$1/2$ degree distributions on the AWGN channel, in terms of E_b/N_0 (dB). The BPSK limit for this rate is 0.1870dB.

max d_v	4	8	9	10	11	12
λ_2	.38354	.30013	.27684	.25105	.23882	.24426
λ_3	.04237	.28395	.28342	.30938	.29515	.25907
λ_4	.57409			.00104	.03261	.01054
λ_5						.05510
λ_6						
λ_7						
λ_8		.41592				.01455
λ_9			.43974			
λ_{10}				.43853		.01275
λ_{11}					.43342	
λ_{12}						.40373
ρ_5	.24123					
ρ_6	.75877	.22919	.01568			
ρ_7		.77081	.85244	.63676	.43011	.25475
ρ_8			.13188	.36324	.56989	.73438
ρ_9						.01087
SNR^*	.8058	.4483	.4090	.3923	.3799	.3727
SNR^*_{GA}	.8459	.5569	.5088	.5135	.5125	.4941
Error	.0401	.1086	.0998	.1212	.1326	.1214

Table 4.2: Exact (SNR*) [69] and Gaussian-approximation (SNR$^*_{GA}$) thresholds for various rate-$1/2$ degree distributions on the AWGN channel, in terms of E_b/N_0 (dB). The BPSK limit for this rate is 0.1870 dB.

max d_v	15	20	30	50
λ_2	.23802	.21991	.19606	.17120
λ_3	.20997	.23328	.24039	.21053
λ_4	.03492	.02058		.00273
λ_5	.12015			
λ_6		.08543	.00228	
λ_7	.01587	.06540	.05516	.00009
λ_8		.04767	.16602	.15269
λ_9		.01912	.04088	.09227
λ_{10}			.01064	.02802
λ_{14}	.00480			
λ_{15}	.37627			.01206
λ_{19}		.08064		
λ_{20}		.22798		
λ_{28}			.00221	
λ_{30}			.28636	.07212
λ_{50}				.25830
ρ_8	.98013	.64584	.00749	
ρ_9	.01987	.34747	.99101	.33620
ρ_{10}		.00399	.00150	.08883
ρ_{11}				.57497
SNR*	.3347	.3104	.2735	.2484
SNR$^*_{GA}$	.5006	.4822	.4629	.4245
Error	.1659	.1718	.1894	.1761

4.3.3 *Numerical methods for density evolution.

Density evolution is a powerful tool for comparing the performance bounds of code ensembles. It may also be used to compare performance bounds of non-ideal or suboptimal algorithms. With these algorithms, the analysis of Section 4.3.1 may be inappropriate.

In this Section, we present an alternative Monte Carlo approach to density evolution. We still employ the Gaussian message approximation, but replace the formal node update rules with "black-box" functions. We denote these functions by $f_v(\underline{U})$ and $f_c(\underline{V})$, for variable and check nodes, respectively. The symbols $\underline{U}$ and $\underline{V}$ represent sets of messages from check and variable nodes, respectively, after the previous iteration.

The quantities of interest are r and $h(s,r)$, the average p_1 message at the input and output of an iteration. The mean input r is given. The mean output can be computed through a sequence of integrations:

$$\begin{array}{ccccccc} r & \longrightarrow & \int f_c(\underline{V})\cdot\rho_V(\underline{V})\cdot d\underline{V} & \longrightarrow & \int f_v(\underline{U})\cdot\rho_U(\underline{U})\cdot d\underline{U} & = & h(s,r) \\ \mu_v & & \text{compute } \mu_u & & \text{compute } \mu_{\text{out}} & & \end{array} \tag{4.3.21}$$

where ρ_V and ρ_U are the probability density functions of variable and check messages, respectively, and μ_v and μ_u are the corresponding mean values. We assume that ρ_v and ρ_u are Gaussian densities obeying the symmetry conditions (4.3.2) and (4.3.3).

The integrals in (4.3.21) can be computed through a variety of numerical methods. In this Section we focus on the use of Monte Carlo integration. The Vegas algorithm [50] is a method of adaptive Monte Carlo integration which uses a combination of importance sampling and stratified sampling to provide a fast and accurate estimate of multi-dimensional integrals. An implementation of the Vegas algorithm is provided as part of the GNU Scientific Library [1], which is used to compute the results presented in this chapter.

Using the Vegas algorithm, an estimate of the integral is quickly obtained to within a specified error tolerance of ±tol. Each estimate $\widehat{x}_i$ thus has an associated error ε_i which is assumed to be a Gaussian random variable with zero mean, and for which $\sigma_{\varepsilon i} = \text{tol}$.

The maximum point of $h(s,r) - r$ can be identified by measuring the curve's slope. Suppose we use the central difference method to estimate the slope at r_m. We choose two points r_1 and r_2 such that r_m is their midpoint. Let $y_1 = h(s,r_1) - r_1$ and $y_2 = h(s,r_2) - r_2$. The slope is then simply $(y_1 - y_2)/(r_1 - r_2) = (y_1 - y_2)/\Delta r$. Let σ_y refer to the standard deviation of error for $h(s,r)$. Then the standard deviation of the slope error, σ_{slope}, is

$$\sigma_{\text{slope}} = \frac{\sqrt{2}}{\Delta r}\sigma_y, \tag{4.3.22}$$

where the $\sqrt{2}$ factor arises because $(y_1 - y_2)$ is the sum of two independent, identically distributed Gaussian random variables.

The resulting error in slope measurements results in a difficult trade-off. If the points are too close together, then σ_{slope} will blow up. For Δr as small as 0.14, σ_{slope} is an order of magnitude larger than σ_y. It is also not possible to make Δr very large, because the curve is not symmetric about the maximum (see Figure 4.3.2).

To see the effect that this has on the estimate of r_m, we model $h(s,r) - r$ locally as a parabola about r_m. Thus $y = y_m - \alpha(r - r_m)^2$, where α is a width parameter. The estimate of r_m, written $\widehat{r_m}$, is equal to r_m plus a Gaussian error term, ε_r. The slope of $h(s,r) - r$ is

$$\begin{aligned} \frac{\partial y}{\partial r} &= -2\alpha(r - r_m) \\ \Rightarrow \varepsilon_{\text{slope}} &= 2\alpha\varepsilon_r \\ \Rightarrow \sigma_r &= \frac{\sigma_y\sqrt{2}}{2\alpha\Delta r}. \end{aligned} \tag{4,3.23}$$

The width parameter, α, must be measured for each code ensemble. For a particular set of degree distributions, α appears (based on observations) to stay constant (or nearly constant) when s is varied.

We now know the variance of our estimates of r_m and the slope at r_m. The next quantity of interest is y_m and its corresponding estimation error, ε_m. We evaluate $h(s,r) - r$ at $r = \widehat{r_m}$. Then

$$\begin{aligned} \widehat{y_m} &= y_m - \alpha(\widehat{r_m} - r_m)^2 + \varepsilon_y \\ \Rightarrow \varepsilon_m &= -\alpha\varepsilon_r^2 + \varepsilon_y \end{aligned} \tag{4.3.24}$$

where the ε_y term arises from the Monte Carlo integration, and the $\alpha\varepsilon_r^2$ term is due to the error in $\widehat{r}_m$.

The error now has a component proportional to the square of a Normal random variable. The resulting error, ε_m, has a systematic bias, because the mean of ε_r^2 is equal to σ_r^2. It is also well known that the variance of ε_r^2 is $2 \cdot \sigma_r^4$. The mean and variance of ε_m are therefore equal to

$$\begin{aligned} \mu_m &= -\alpha \cdot \sigma_r^2 \\ &= -\left(\frac{1}{2\alpha}\right) \cdot \left(\frac{\sigma_y}{\Delta r}\right)^2, \qquad (4.3.25) \\ \sigma_m^2 &= 2 \cdot \alpha^2 \cdot \sigma_r^2 + \sigma_y^2 \\ &= 2\alpha\,|\mu_m| + \sigma_y^2. \qquad (4.3.26) \end{aligned}$$

As long as $\sigma_y/\Delta r \ll 1$, the bias μ_m is very small, and $\sigma_m \approx \sigma_y$.

Finally, we estimate the threshold, s^*, using a sequence of $\widehat{y_m}$ estimates. When $\widehat{y_m}(s)$ is sufficiently close to zero, we terminate the search. We wish to evaluate the error in this estimate. Let β be the slope of $y_m(s)$, evaluated at s^*:

$$\beta \equiv -\left.\frac{\partial y_m}{\partial s}\right|_{s=s^*} \qquad (4.3.27)$$

Then the threshold error, ε_s, is approximated by

$$\varepsilon_s = \frac{\varepsilon_m}{\beta}. \qquad (4.3.28)$$

This estimate is also biased. The mean and standard deviation of the threshold measurement are equal to those of $\widehat{y_m}$, scaled by $1/\beta$:

$$\mu_s = -\left(\frac{1}{2\alpha\beta}\right) \cdot \left(\frac{\sigma_y}{\Delta r}\right)^2 \qquad (4.3.29)$$

$$\sigma_s = \frac{1}{\beta}\sqrt{2 \cdot |\mu_s|^2 + \sigma_y^2} \approx \frac{\sigma_y}{\beta}. \qquad (4.3.30)$$

The accuracy of the threshold estimate is now specified in terms of known or measurable parameters. The parameters α and β are typically greater than 1. The critical choice is the ratio $\sigma_y/\Delta r$, which should be on the order of 0.01 for good estimation.

Table 4.3: Estimated parameters, bias (μ_s), error standard deviation (σ_s), and threshold results for various regular LDPC ensembles. In all cases, the tolerance is $\sigma_y = 0.001$ and $\Delta r = 0.025$.

d_v	d_c	r	α	β	μ_s	σ_s	$\widehat{s^*}$	$\left(\frac{E_b}{N_0}\right)^*$ dB	BPSK Limit (dB)
3	4	0.25	1.09	30.5	$2.405 \cdot 10^{-5}$	$3.28 \cdot 10^{-5}$	0.3012	0.8091	−0.762
3	5	0.4	1.88	32.6	$1.305 \cdot 10^{-5}$	$3.07 \cdot 10^{-5}$	0.4755	0.7509	−0.203
3	6	0.5	2.45	35.7	$9.15 \cdot 10^{-6}$	$2.80 \cdot 10^{-5}$	0.6297	1.002	0.202
3	8	0.625	3.53	40.0	$5.67 \cdot 10^{-6}$	$2.50 \cdot 10^{-5}$	0.885	1.511	0.774
3	9	2/3	4.01	41.9	$4.76 \cdot 10^{-6}$	$2.39 \cdot 10^{-5}$	0.9935	1.733	0.992
3	12	3/4	5.32	46.2	$3.25 \cdot 10^{-6}$	$2.16 \cdot 10^{-5}$	1.258	2.246	1.50
4	5	0.2	1.90	25.8	$1.63 \cdot 10^{-5}$	$3.88 \cdot 10^{-5}$	0.343	2.34	−0.940
4	6	1/3	2.59	28.2	$1.095 \cdot 10^{-5}$	$3.55 \cdot 10^{-5}$	0.4704	1.50	−0.457
4	8	0.5	3.60	32.6	$6.82 \cdot 10^{-6}$	$3.07 \cdot 10^{-5}$	0.6931	1.42	0.202
4	9	0.556	4.09	34.3	$5.70 \cdot 10^{-6}$	$2.92 \cdot 10^{-5}$	0.7886	1.52	0.444
4	10	0.6	4.61	35.7	$4.86 \cdot 10^{-6}$	$2.80 \cdot 10^{-5}$	0.8737	1.63	0.651
4	12	2/3	5.41	38.5	$3.84 \cdot 10^{-6}$	$2.60 \cdot 10^{-5}$	1.027	1.88	0.992

Example 4.3.1. : For a (3,4) regular LDPC ensemble, a set of estimation parameters are chosen and/or measured as follows:

$$\begin{aligned} \sigma_y &= 0.001 \\ \Delta r &= 0.025 \\ \alpha &\approx 1.09 \\ \beta &\approx 30.5. \end{aligned}$$

This results in an estimation error with these characteristics:

$$\begin{aligned} \mu_s &= 2.405 \cdot 10^{-5} \\ \sigma_s &= 3.279 \cdot 10^{-5}. \end{aligned}$$

The calculated threshold for this ensemble is 0.3012. Results for several regular ensembles are presented in Table 4.3.

□

4.3.4 *Iterated integration.

The method proposed in Section 4.3.3 requires iterated Monte Carlo integration. Each integration is an estimate with its own error. While a particular estimate may have an error with variance σ_{MC}^2, the final estimate will have a greater error variance, because of the propagation of intermediate estimation errors.

Let $\mathcal{M}_u(\mu_v)$ be the integral estimator for check nodes, and $\mathcal{M}_v(\mu_u)$ the estimator for variable nodes. In the first estimation, we have

$$\widehat{\mu_u} = \mathcal{M}_u(\mu_v) = \mu_u + \varepsilon_u. \tag{4.3.31}$$

Then in the second estimation,

$$\begin{aligned} \widehat{\mu_{\text{out}}} &= \mathcal{M}_v(\mu_u + \varepsilon_u) \\ &\approx \mu_{\text{out}} + \varepsilon_v + \chi\varepsilon_u, \\ \text{where } \chi &\equiv \left.\frac{\partial\mu_{\text{out}}(\mu)}{\partial\mu}\right|_{\mu=\mu_u}. \end{aligned} \tag{4.3.32}$$

The overall error ε_{out} has $\sigma_{\text{out}} = \sigma\sqrt{1+\chi}$, where σ corresponds to σ_y in Section 4.3.3.

To properly account for the error in μ_{out}, it is necessary to compute an estimate of χ, which is given by

$$\begin{aligned} \chi &= \frac{\partial}{\partial\mu}\int f(\underline{x})\,\rho(\mu,\underline{x})\,d\underline{x} \\ &= \int f(\underline{x})\frac{\partial}{\partial\mu}\left[\rho(\mu,\underline{x})\right]d\underline{x} \end{aligned} \tag{4.3.33}$$

where $\rho(\mu,\underline{x})$ is the product of i.i.d. Gaussian LLR density functions with mean μ, and $\underline{x}$ is a vector of input messages. After evaluating the derivative, we find that (for one dimension)

$$\frac{\partial}{\partial\mu}\left[\rho(\mu,x)\right] = \rho(\mu,x)\left[\left(\frac{x-\mu}{4\mu}\right)\left(2+\frac{x-\mu}{\mu}\right)-\frac{1}{2\mu}\right]. \tag{4.3.34}$$

When iterated integration is used, the parameter σ_y in Section 4.3.3 must be replaced by σ_{out}, which is itself an estimate and must be computed numerically using (4.3.33) and (4.3.34).

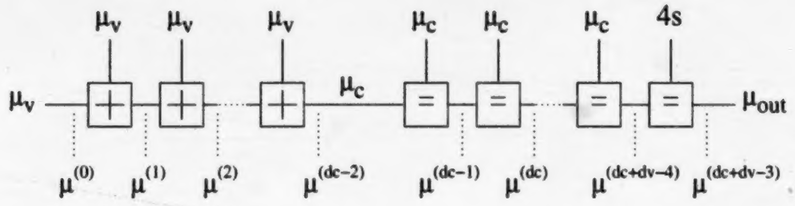


Figure 4.3.3: Iterated integration using only three-edge nodes.

It may also be necessary to generalize this error analysis to account for many iterated integral estimations. Such a scenario is illustrated in Figure 4.3.3, in which check and equality nodes are treated as a cascade of three-edge nodes. This reduces the number of dimensions in the integrals of (4.3.21), and may make the computation more efficient.

As indicated in Figure 4.3.3, this approach requires $d_c + d_v - 3$ integrations. The corresponding exact integrals are labeled $\mu^{(i)}$. An iterated sequence of Monte Carlo integrations results in a sequence of estimations $\widehat{\mu^{(i)}}$, defined recursively as

$$\begin{aligned} \widehat{\mu^{(i)}} &= \mu^{(i)} + \varepsilon_i + \varepsilon_{i-1} \cdot \chi_i \left(\mu^{(i-1)}\right), \; i > 0 \\ \text{where } \chi_1 &= 0. \end{aligned} \tag{4.3.35}$$

Note that each χ_i is a function of $\mu^{(i-1)}$, and $\mu^{(0)}$ is the input. The resulting standard deviation of the estimate is also a recursive sequence:

$$\sigma_i = \sqrt{\sigma_y^2 + \sigma_{i-1}^2 \chi_i^2}. \tag{4.3.36}$$

The final error deviation calculated by (4.3.36) is the actual value of σ_y which should be used in all expressions in Section 4.3.3.

4.4 An algorithm for computing the threshold.

In this section we present a practical algorithm for carrying out density evolution with the Gaussian approximation. The algorithm is simple, and is not presented as an optimal solution. It is presented in order to provide a tangible demonstration of density evolution.

As in previous sections, we use the notation $y(r) = h(s,r) - r$. The algorithm is divided into two phases. During the first phase, we ensure that the maximum is large, and a coarse search with low precision is used to locate it. We then improve the precision and find the point through small adjustments.

1. Initialize $s = s_0$ to a low value, perhaps near the BPSK Limit (Section 2.2). This ensures that the maximum of y, $y_m^{(0)}$, is greater than zero and easy to find.
2. Allow a high integration error tolerance, and compute $y(r)$ for a sequence of r values, beginning with r slightly greater than zero. Increase r in small increments, measuring the slope. Stop when the slope crosses zero. Call this point r_0.
3. Now switch to a low integration error tolerance. Choose two points r^+ and r^- such that $r^- < r_0 < r^+$ and $r^+ - r^- = \Delta r$. Measure the slope at r_0 using the formula $y'(r_0) = (y^+ - y^-)/\Delta r$. Shift r_0 in decreasing increments until the slope is sufficiently close to zero. Call the resulting point $r_m^{(0)}$.

Having measured $r_m^{(0)}$, α is estimated according to

$$\alpha \approx \frac{y'(r)}{2\left(r - r_m^{(0)}\right)} \tag{4.4.1}$$

for some point r near $r_m^{(0)}$.

In the next phase of the algorithm, we increase s slightly so that $s = s_1 > s_0$, and use $r_m^{(0)}$ as an initial guess for the new maximum of $h(s_1,r) - r$. We wish to estimate the new maximum point $\left(r_m^{(1)}, y_m^{(1)}\right)$. To do this, we assume that $y(r)$ is a parabola in the neighborhood of $r_m^{(1)}$, and we assume that $r_m^{(0)}$ is close to $r_m^{(1)}$. We find the new maximum as follows:

1. Let $r_0 = r_m^{(0)}$. Compute $y'(r_0)$ using the central difference method.
2. The maximum r_m is approximately located at

$$r_m \approx r_0 + \frac{y'(r_0)}{2\alpha}. \tag{4.4.2}$$

We use this formula to generate a sequence of guesses $r_i = r_{i-1} + \frac{y'(r_{i-1})}{2\alpha}$.

3. We stop when $y'(r_i)$ is sufficiently close to zero, and declare $r_m^{(1)} = r_i$ and $y_m^{(1)} = y(r_i)$.

Once a pair of maximum points has been observed, we select the next threshold guess, s_2, by using a linear approximation. We arrive at a sequence of guesses using the update rule:

$$\beta_j = \frac{y_m^{(j)} - y_m^{(j-1)}}{s_j - s_{j-1}}, \tag{4.4.3}$$

$$s_{j+1} = s_j - \frac{y_m^{(j)}}{\beta_j}. \tag{4.4.4}$$

This procedure is repeated until $y_m^{(j)}$ is sufficiently close to zero. We then declare $s^* \approx s_j$.

A typical sequence of $y(r)$ estimates computed during execution of this algorithm is shown in Figure 4.4.1. Both the low-precision and high-precision estimations are represented, by dashed and solid curves, respectively. The corresponding sequence of measured slopes, $y'(r_i)$, is shown in Figure 4.4.2. These curves represent the algorithm's execution for a $(4, 12)$ regular ensemble.

In Figure 4.4.2, the fine-precision search begins as soon as a zero-slope is detected. Each time s is altered, the slope jumps away from zero, but is immediately corrected after application of the update rule (4.4.2).

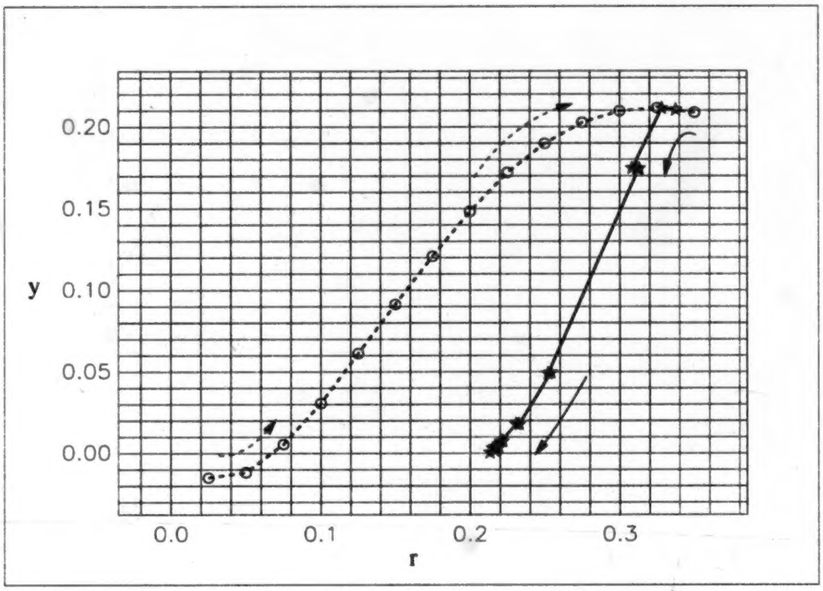


Figure 4.4.1: A sequence of estimations of $h(s,r) - r$ during a threshold search. The initial low-precision search is shown as a dashed curve. The sequence of high-precision estimates of $y_m^{(j)}$ is joined by a solid curve.

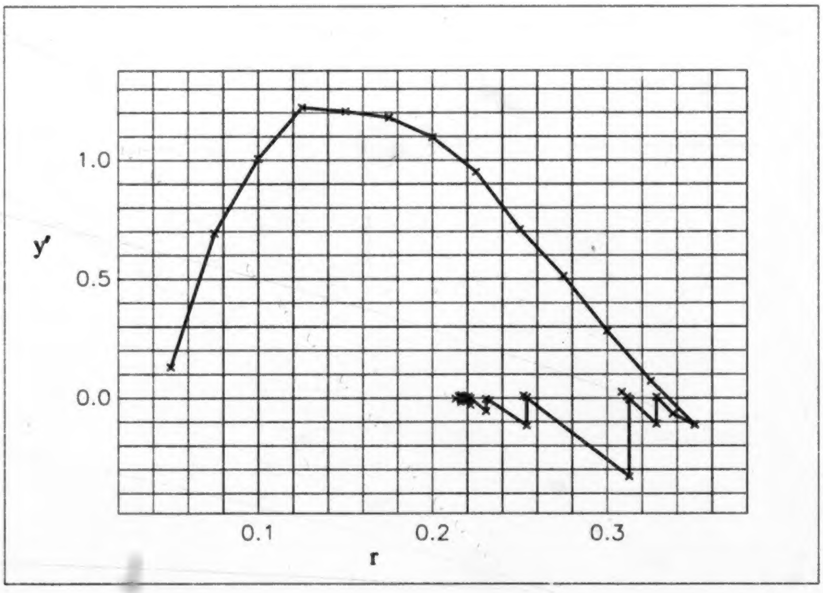


Figure 4.4.2: A sequence of slopes measured during threshold search.

No text

Chapter 5

Analog Decoding Circuits

5.1 The translinear principle.

In order to implement the sum-product algorithm, we need two operations: multiplication and addition. A third operation, normalization, is also helpful in a real implementation. Normalization of a vector $\underline{x}$, written $|\underline{x}|$, is defined by

$$|\underline{x}| \equiv \frac{\underline{x}}{\sum_{i=1}^{n} x_i}. \tag{5.1.1}$$

As we mentioned in Section 2.4.2, normalization of probability masses does not affect the correctness of the algorithm.

These operations are conveniently provided by a class of circuits known as *translinear circuits* [75]. A three-terminal *translinear device*, illustrated in Figure 5.1.1, is basically an idealized transistor.

For the purposes of this thesis, we say that a translinear device has three ports: the gate, the source and the drain. The drain current I_d flows between the drain and the source, and is controlled by the voltage v_{gs} between the gate and the source.

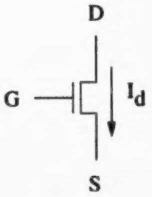


Figure 5.1.1: A three-terminal translinear device.

The idealized relationship between I_d and v_{gs} is

$$I_d = I_0 e^{n \cdot v_{gs}}, \tag{5.1.2}$$

where I_0 and n are proportionality constants with units of A and V^{-1}, respectively.

Such a device is called "translinear" because its transconductance, defined as

$$g_m \equiv \left. \frac{\partial I_d}{\partial v_{gs}} \right|_{I_d = I_{d0}}, \tag{5.1.3}$$

is linear with respect to a fixed bias current I_{d0}. In the case of a device modeled by (5.1.2), the transconductance is $g_m = n \cdot I_{d0}$. We will see that MOS transistors, biased in their subthreshold operating region, can be used as translinear devices. Before exploring the details of physical MOS transistors, we introduce fundamental translinear building blocks using the very simple model (5.1.2).

Translinear devices can be used for multiplication of analog currents because they can be arranged in *translinear loops*. A translinear loop is a closed Kirchoff voltage loop which traverses the v_{gs} of an even number of translinear devices. An example of such a loop is shown in Figure 5.1.2. Note that the bottom two transistors are upside-down in this figure. By tracing a closed loop across devices 1, ..., 4, we arrive at

$$\begin{aligned} v_{gs1} + v_{gs3} &= v_{gs2} + v_{gs4} \\ \Rightarrow \ln\left(\frac{I_1}{I_0}\right) + \ln\left(\frac{I_3}{I_0}\right) &= \ln\left(\frac{I_2}{I_0}\right) + \ln\left(\frac{I_4}{I_0}\right) \\ \Rightarrow I_1 \cdot I_3 &= I_2 \cdot I_4. \end{aligned} \tag{5.1.4}$$

Because of the orientation of voltages in the loop, we refer to currents on the left-hand side of (5.1.4) as *clockwise currents*, and those on the right-hand side as *counter-clockwise currents*. It is a simple exercise to generalize (5.1.2), yielding the following principle:

Definition 5.1.1. The Translinear Principle. In a translinear loop, the product of clockwise currents is equal to the product of counter-clockwise currents.

□

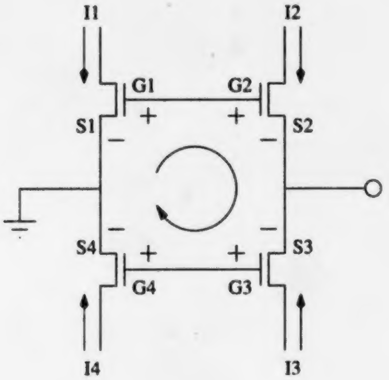


Figure 5.1.2: A simple translinear loop.

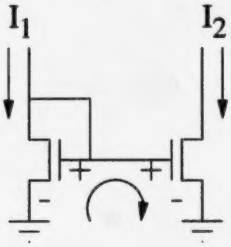


Figure 5.1.3: Current mirror circuit.

5.1.1 Basic translinear circuits.

We now apply the translinear principle to some important basic circuits. The simplest is the current mirror, shown in Figure 5.1.3. In this circuit, I_1 and I_2 have opposite orientation. They are the only two currents in the loop, so $I_1 = I_2$.

Another simple case is the Gilbert multiplier, shown in Figure 5.1.4, in which we assume that Ix_1, Ix_2 and Iy_1 are known (inputs), while Iz_1 and Iz_2 are unknown (outputs). This circuit contains two distinct translinear loops. We recognize loop 1 as a current mirror, so that $I_S = Iy_1$. Loop 2 traverses from node A to node B, and then back to node A. The translinear principle dictates that

$$Ix_1 \cdot Iz_2 = Ix_2 \cdot Iz_1. \tag{5.1.5}$$

Because current only flows from the drain to the source of a translinear device, it is

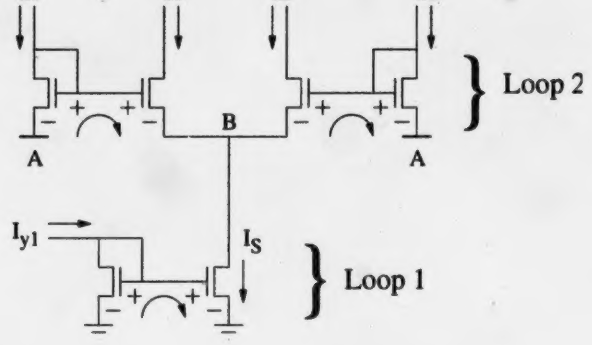


Figure 5.1.4: A simple Gilbert multiplier.

also the case that

$$
\begin{aligned}
I_S &= Iz_1 + Iz_2 \\
\Rightarrow Iy_1 &= Iz_1 \cdot \left(1 + \frac{Ix_2}{Ix_1}\right) \\
\Rightarrow Iz_1 &= \frac{Iy_1 \cdot Ix_1}{Ix_1 + Ix_2}.
\end{aligned}
\tag{5.1.6}
$$

The simple Gilbert multiplier circuit thus provides a *normalized multiplication* (in the positive quadrant) of analog currents. This takes care of two of the necessary operations for sum-product decoding. The third operation, *addition* of analog currents, is as simple as shorting wires together.

We may also expand the multiplier of Figure 5.1.4 by adding an arbitrary number of Ix inputs, as illustrated in Figure 5.1.5. Based on the derivation of (5.1.6), it is easy to verify that

$$
Iz_{ij} = \frac{Iy_j \cdot Ix_i}{\sum_{k=1}^{n} Ix_k} \tag{5.1.7}
$$

$$
\underline{z} = Iy_j \cdot |\underline{x}|. \tag{5.1.8}
$$

The circuit of Figure 5.1.5 therefore multiplies a vector of currents, $\langle Ix_i \rangle_{i=1}^{N}$, by a scalar Iy_j, while normalizing to ensure that $\sum_i Iz_i = Iy_j$. We could also describe (5.1.7) as *normalization* of a current vector. This normalizing operation is linear, so that for any pair i, j, $Ix_i / Ix_j = Iz_i / Iz_j$.

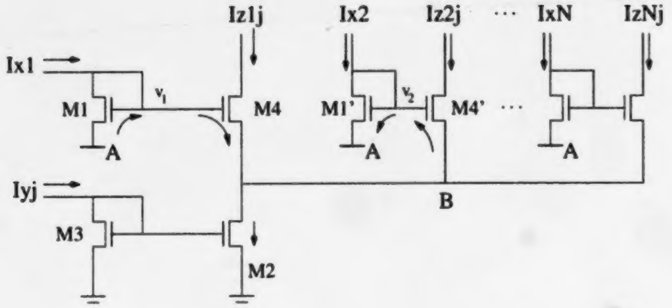


Figure 5.1.5: A Gilbert multiplier circuit for vector scaling.

5.1.2 The Gilbert vector multiplier.

The simple multiplier of Figure 5.1.4 can be generalized to provide normalized vector and matrix multiplications [52]. This circuit is shown in Figure 5.1.6, where again we assume that Ix_i and Iy_j are known inputs, while Iz_{ij} are unknown outputs. The circuit of Figure 5.1.6 is simply a repetition of the basic Gilbert multiplier of Figure 5.1.4. The current mirrors at the bottom have been replaced with current sources to simplify the diagram.

The translinear loops in this circuit are replications of the loops from Figure 5.1.4. Applying the same analysis, we find that for $1 \le i \le n$, $1 \le k \le n$, and $1 \le j \le m$,

$$Iz_{ij} \cdot Ix_k = Iz_{kj} \cdot Ix_i. \tag{5.1.9}$$

We also find that for $1 \le j \le m$,

$$\sum_{k=1}^{n} Iz_{kj} = Iy_j. \tag{5.1.10}$$

Equations 5.1.9 and 5.1.10 provide enough information to solve for the output Iz_{ij}:

$$\begin{aligned} Iy_j &= \frac{Iz_{ij}}{Ix_i} \cdot \sum_{k=1}^{n} Ix_k \\ \Rightarrow Iz_{ij} &= \frac{Iy_j \cdot Ix_i}{\sum_{k=1}^{n} Ix_k} \end{aligned} \tag{5.1.11}$$

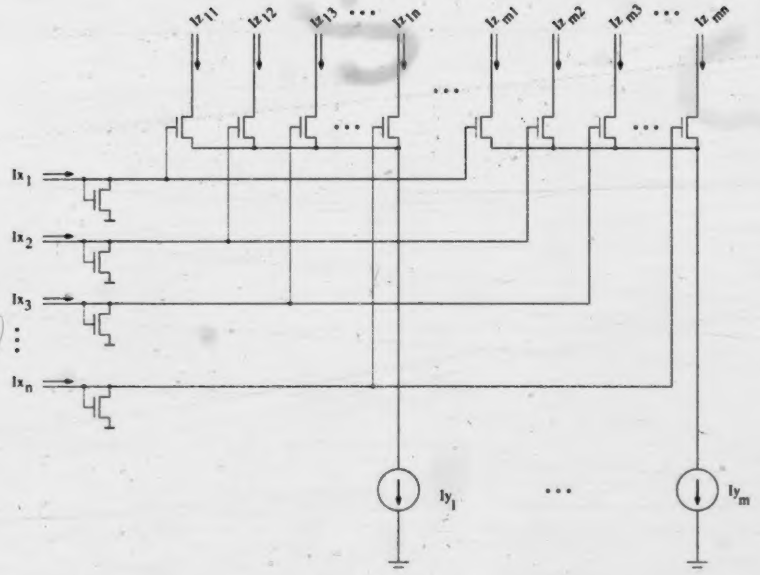


Figure 5.1.6: Gilbert vector multiplier.

We can also represent (5.1.11) in terms of vector and matrix operations. Let $\underline{x}$ and $\underline{y}$ be *column* vectors whose elements are the currents $\{Ix_1, ..., Ix_n\}$ and $\{Iy_1, ..., Iy_m\}$, respectively. Let Z be an $n \times m$ matrix whose $(i, j)^{\text{th}}$ member is equal to Iz_{ij}. Then

$$Z = \frac{\underline{y}\,\underline{x}^{\mathrm{T}}}{\sum_k x_k} = \underline{y}\,|\underline{x}|^{\mathrm{T}}. \tag{5.1.12}$$

5.1.3 Translinear sum-product circuits.

Suppose we design a Gilbert multiplier circuit so that $\underline{x}$ and $\underline{y}$ are proportional to *probability masses*. Let I_U be an arbitrary current which represents a probability of one. Suppose $\mathbf{x}$ is a discrete random variable with n possible outcomes $\mathcal{A}_{\mathbf{x}} = \{x_1, ..., x_n\}$. Then we say that $\underline{x}$ represents a probability mass for $\mathbf{x}$ if $Ix_i = I_U \cdot \Pr\{\mathbf{x} = x_i\}$.

If we use the normalized current unit $A' \equiv I_U$, so that $I_U = 1A'$, then the denominator of (5.1.12) becomes $\sum_k x_k = 1A'$, and can be neglected. From this perspective, the Gilbert vector multiplier supplies every multiplication between the probability masses of $\mathbf{x}$ and $\mathbf{y}$. Because the products are output as analog currents, summation

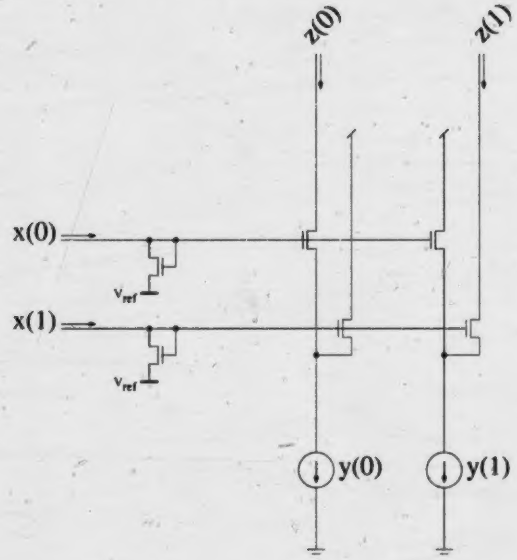


Figure 5.1.7: A Gilbert-multiplier implementation of the equality node.

is as simple as shorting wires. The Gilbert multiplier therefore provides the core operations needed to implement the sum-product algorithm expressed by (2.4.3).

Example 5.1.2. Equality circuit. A translinear implementation of the SP algorithm for an equality node is shown in Figure 5.1.7. For this simple node, we only need the products $z_0 = x_0 \cdot y_0$ and $z_1 = x_1 \cdot y_1$. The Gilbert multiplier produces those products, as well as the unwanted products $x_0 \cdot y_1$ and $x_1 \cdot y_0$, which are simply discarded by tying them to V_{dd}.

□

5.1.4 Duality in translinear circuits.

In Section 2.1.1 we examined the duality between log-likelihood ratios and probability masses. The same duality exists naturally in translinear sum-product circuits. When analog currents are used to represent probabilities, *differential voltages* are proportional to log-likelihood ratios.

This duality is demonstrated with the differential pair circuit of Figure 5.1.8.

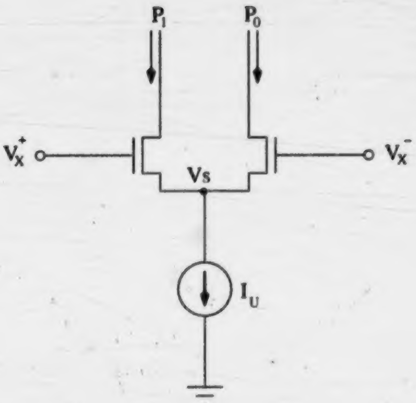


Figure 5.1.8: Differential pair circuit.

Let $V_x = V_x^+ - V_x^-$. The currents P_0 and P_1 are given by

$$P_0 = e^{(V_x^- - V_s)} \tag{5.1.13}$$

$$P_1 = e^{(V_x^+ - V_s)}. \tag{5.1.14}$$

After a simple rearrangement, we see that

$$\begin{aligned} \ln\left(\frac{P_1}{P_0}\right) &= \ln\left(e^{(V_x^+ - V_s - V_x^- + V_s)}\right) \\ &= V_x, \end{aligned}$$

therefore V_x is proportional to a log-likelihood ratio. Every translinear sum-product circuit is thus equally well described using the language of probabilities (current) or that of log-likelihood ratios (voltage).

5.2 CMOS translinear circuits.

Translinear devices are idealized circuit elements. As it turns out, MOS devices ("Metal-Oxide-Semiconductor") exhibit translinear behavior when their current bias is very small (typically less than $1\,\mu A$). This region of operation is often referred to as *subthreshold* or *weak-inversion*. In this section, we re-examine the translinear circuits of Section 5.1, specialized to MOS devices.

There are two types of MOS devices, *positive* channel (PMOS) and *negative* channel (NMOS). These are actually four-terminal devices, as indicated in Figure

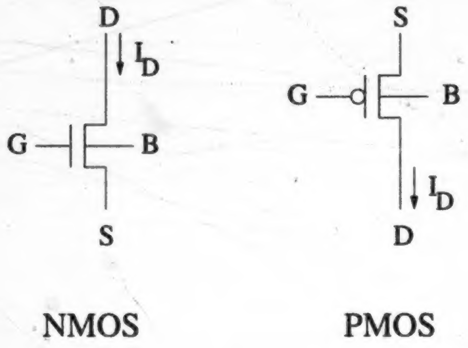


Figure 5.2.1: Complementary MOS devices.

5.2.1. Both types of devices have gate, drain, source, and *bulk* (also known as the *body*) terminals. Note that the source and drain terminals are reversed in the PMOS device. The *drain current* I_D flows between the source and drain terminals. Approximately no current flows through the gate of either device (the gates act as capacitors), and approximately no net current flows through the base.

For NMOS devices, the base is usually connected to the most negative potential in the circuit, labeled V_{SS} (or simply *ground* if the circuit has a single-sided power supply). For PMOS devices, the base is almost always connected to the most positive potential in the circuit, labeled V_{dd}. For most "first-order" circuit analysis, the base terminal is ignored. It is only relevant when the source-to-base voltage, v_{sb}, is non-zero.

In current-mode circuit design, it is helpful to distinguish between devices which act as *current sources*, and those which act as *loads*. For the most part, any device in which the gate and drain terminals are connected is a load. This is called a *diode-connected* configuration. For the most part, any other device is a current source. In translinear circuits, current inputs arrive at loads, and outputs are drawn from current sources. Any PMOS current source must be connected to an NMOS load, and any NMOS current source must be connected to a PMOS load. This is illustrated in Figure 5.2.2.

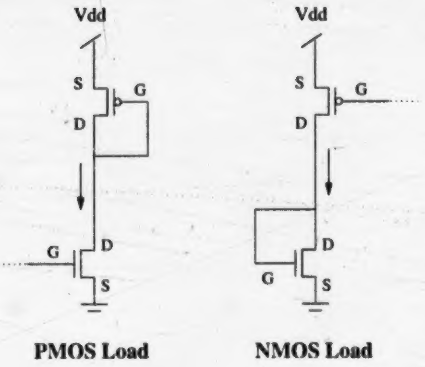


Figure 5.2.2: Configuration of devices as current sources and diode-connected current loads.

5.2.1 MOS device models.

In conventional digital design, MOS devices are treated like switches. Device operation is governed with respect to a threshold voltage V_{th}. If $|v_{gs}| > V_{\mathrm{th}}$, then the device is *on* (closed circuit). Otherwise, it is *off* (open circuit). A more detailed standard model of MOS devices is the so-called *square law*. The square law dictates that the device current, under certain operating conditions, is proportional to $(v_{gs} - V_{\mathrm{th}})^2$.

The square law applies when the drain-source voltage, v_{ds}, is greater than $v_{gs} - V_{\mathrm{th}}$, and when the transistor is in *strong inversion*. Usually, strong inversion is where $v_{gs} > V_{\mathrm{th}}$, as opposed to *weak inversion* or *subthreshold* operation, where $v_{gs} < V_{\mathrm{th}}$.

Definition 5.2.1. Strong-inversion. When $v_{gs} > V_{\mathrm{th}}$, the transistor is said to operate in strong-inversion, which refers to the presence of inverted charge carriers within the device's channel. In strong inversion, the physical mechanism of current flow in the device is dominated by drift current.

If the drain-source voltage satisfies $v_{ds} > v_{gs} - V_{\mathrm{th}}$, and the device is operating in strong inversion, then

$$I_D = \frac{1}{2}\mu \cdot C'_{\mathrm{ox}} \cdot \frac{W}{L} \cdot \left(|v_{gs}| - V_{\mathrm{th}}\right)^2. \tag{5.2.1}$$

where μ and C'_{ox} are fixed parameters of the fabrication process. W and L are the width and length of the MOS device, respectively.

□

When v_{gs} is below the threshold, a very small device current flows. In conventional CMOS circuit design, this subthreshold current is often said to be effectively zero. In fact, the very small device currents in this region allow us to construct MOS translinear circuits which provide analog computing functions with extremely low power consumption.

Definition 5.2.2. Weak Inversion. When $v_{gs} < V_{\text{th}}$, and when the device current, I_D, is less than one-tenth of the device's *specific current*, I_S, the device is said to operate in weak inversion. In weak inversion, the dominant mechanism of current flow is diffusion.

The specific current is defined as

$$I_S \equiv \frac{2\mu C_{\text{ox}} U_T^2 W}{\kappa L} \cdot \exp\left(-\frac{\kappa V_{T0}}{U_T}\right) \tag{5.2.2}$$

where μ is the mobility of charge carriers in the device, C_{ox} is the oxide capacitance, $U_T \approx 0.025$ V is the well-known thermal voltage, W and L are the device's channel width and length, respectively, and κ and V_{T0} are constants of the fabrication process. Typically $\kappa \approx 0.7$ (unitless), but κ may lie in a range between 0.5 and 0.99 [58]. V_{T0} is closely related to the threshold voltage, and has units of Volts. V_{T0} is usually less than 1 V, and may be as low as 0.3 V, depending on the particular fabrication process.

When operating in weak inversion, MOS transistors obey the model

$$\begin{aligned} I_D &= I_S \cdot f_s(v_s) \cdot f_g(v_g) \cdot e^{\frac{\kappa \cdot v_{gs}}{U_T}} \cdot \left[1 - f_{ds}(v_{ds}) \cdot e^{-\frac{\kappa \cdot v_{ds}}{U_T}}\right] && (5.2.3) \\ &\approx I_0 \cdot \exp\left\{\frac{\kappa \cdot v_{gs}}{U_T}\right\} \cdot \left[1 - \exp\left\{-\frac{\kappa \cdot v_{ds}}{U_T}\right\}\right] && (5.2.4) \end{aligned}$$

where f_s, f_g, and f_{ds} account for small non-linear fluctuations in the device model. It is often sufficient to use the approximate model (5.2.4), where we consider I_0 to be a small constant current [82].

Circuits based on this subthreshold model were popularized by Vittoz, et. al. [82] and Mead [58].

□

If we write all voltages in units of $\mathrm{V}' \approx \frac{\kappa}{U_T} \cdot \mathrm{V}$, and assume that all devices have equal dimensions, and if we write all currents in units of $\frac{A}{I_0}$, then (5.2.4) becomes

$$I_D = e^{v_{gs}} \cdot \left[1 - e^{-v_{ds}}\right] \tag{5.2.5}$$

Furthermore, if v_{ds} is sufficiently large (e.g. greater than $4U_T \approx 3\mathrm{V}' \approx 200$ mV), then the term $e^{-v_{ds}}$ in (5.2.5) can be neglected.

Definition 5.2.3. Saturation. When an MOS device is operating in weak inversion, and $e^{-v_{ds}}$ is small enough to be neglected, then the device is said to be in saturation. In this region, the device current is governed by

$$I_D = e^{v_{gs}}. \tag{5.2.6}$$

□

We immediately recognize (5.2.6) as the model of a translinear device. In weak-inversion, saturated MOS transistors therefore operate as translinear devices. In effect, this means we can do useful computation even when, from the conventional perspective of operating above V_{th}, all of the transistors in a circuit are *turned off*.

Definition 5.2.4. Unsaturated model. If v_{ds} is not sufficiently large, then the effect of $e^{-v_{ds}}$ must be accounted for in the model. The equation for I_{ds} can then be expressed in terms of forward and reverse components [74]:

$$\begin{aligned} I_D &= I_f - I_r \\ \text{where } I_f &= e^{v_{gs}} \\ \text{and } I_r &= e^{v_{gd}}. \end{aligned} \tag{5.2.7}$$

In (5.2.7) we have used the same normalized current and voltage units as in (5.2.5). This situation can be analyzed using the translinear principle, if the transistor is modeled by a pair of parallel translinear devices. The first device is an ordinary transistor with device current I_f. The second is a "backward" transistor, with device current I_r, controlled by the voltage drop v_{gd}. This model is illustrated in Figure 5.2.3. To apply this model, we incorporate the backward transistor in an additional set of translinear loops.

□

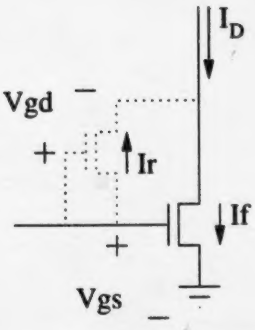


Figure 5.2.3: Unsaturated weak-inversion translinear MOS device.

As a final note on MOS device models, we must address what happens when $v_{gs} \approx V_{\text{th}}$ and $I_D \sim I_S$. In this case, the device is somewhere between weak-inversion and strong inversion.

Definition 5.2.5. Moderate Inversion. When an MOS transistor has a device current satisfying $0.1 < I_D/I_S < 10$, the device is said to operate in moderate inversion. In this region, both drift and diffusion mechanisms make significant contributions to the current flow. The device's behavior is somewhere between the models (5.2.1) and (5.2.3), but is difficult to describe in terms which are both precise and general. □

5.2.2 The canonical CMOS sum-product circuit.

An approach for CMOS analog decoder designs, elaborated in [52], has emerged as a popular topology for analog decoder designs. This topology, which is based on the generalized Gilbert multiplier of Figure 5.1.6, is shown in Figure 5.2.4. Figure 5.2.4 uses a simplified "box notation," explained in Figure 5.2.5, in which an array of source-connected NMOS transistors is indicated by a large box. Because of its relative popularity, the architecture of Figure 5.2.4 will be referred to as the *canonical* topology.

In Figure 5.2.4, X denotes the ordered sequence of *row inputs* (nodes) $< x_1, ..., x_N >$, and Y denotes the ordered sequence of *column inputs* (nodes) $< y_1, ..., y_M >$. The current inputs Ix_i and Iy_k arrive at input nodes x_i and y_k, located on the diode-

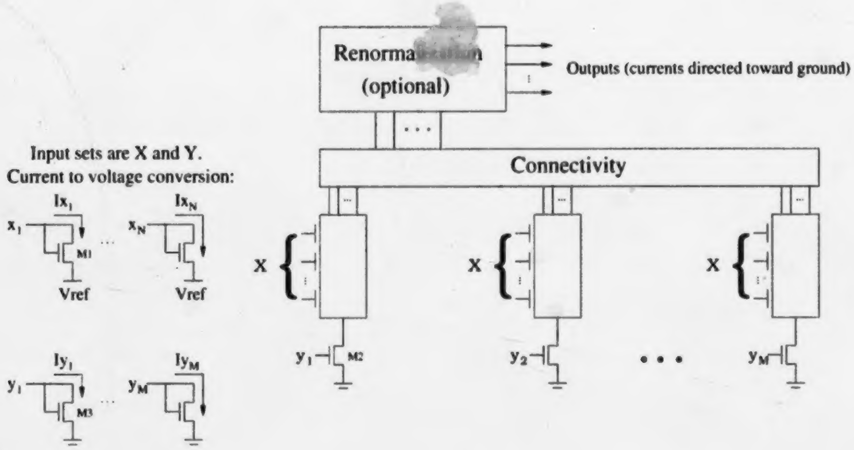

Renormalization
(optional)
Outputs (currents directed toward ground)
Input sets are X and Y.
Current to voltage conversion:
Ix_1
Ix_N
x_1
x_N
M1
Vref
Vref
Iy_1
Iy_M
y_1
y_M
M3
Connectivity
X
X
X
y_1
M2
y_2
y_M

Figure 5.2.4: Canonical sum-product circuit topology.

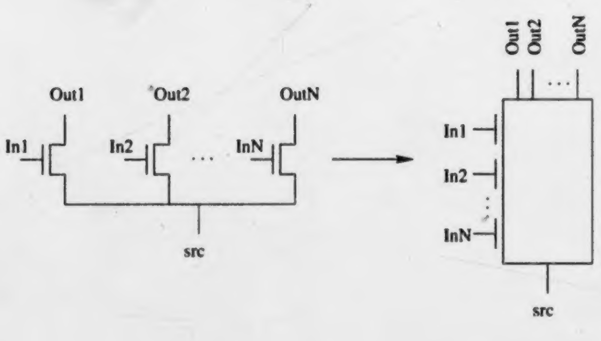

Out1
Out2
OutN
In1
In2
InN
src
Out1
Out2
OutN
In1
In2
InN
src

Figure 5.2.5: Symbol for a source-connected transistor array.

connected transistors on the left, labeled M1 and M3. All devices typically have the same dimensions in a canonical sum-product circuit. Each input Ix_i is derived from the probability $\rho_{\mathbf{x}}(i)$:

$$Ix_i = I_U \cdot \rho_{\mathbf{x}}(i)\,. \tag{5.2.8}$$

Intermediate outputs emerge from the top of the source-connected boxes in Figure 5.2.4. There are $N \times M$ such outputs, referred to as Iz_{ij}, corresponding to one row position i and one column position j. The outputs represent multiplication of row and column inputs. Unused products (those for which the constraint f is not satisfied) must be shorted to V_{dd}.

5.2.3 Translinear analysis

In this section we present an analysis of the canonical MOS sum-product topology based on the translinear principle. Alternative forms of this analysis are elaborated in [52, 53]. In the canonical sum-product topology, all transistors are assumed to be in saturation. A portion of the canonical topology is shown in Figure 5.1.5, in which M2 and M4 must be in saturation. To ensure saturation, a reference voltage $V_{\mathrm{ref}} \approx 0.3\mathrm{V}$ is used at the source of M1 and related devices (labeled node A in Figure 5.1.5). This maintains a sufficiently high voltage at the drain of M2 for M2 to remain in saturation.

We now verify that the translinear principle is accurate for weak inversion MOS circuits, in spite of the extra non-linear factors in the detailed model (5.2.3). This analysis is repeated from [74]. First, we write v_{gs} in terms of I_D using the full model:

$$v_{gs} = \frac{U_T}{\kappa} \ln\left(\frac{I_D}{I_s \cdot f_s(v_s)\, f_g(v_g)}\right). \tag{5.2.9}$$

By taking a closed loop in Figure 5.1.5, beginning and ending at A, and traversing the gate voltages v_1 and v_2, we find that

$$\begin{array}{rcl} (v_1 - A) + (v_2 - B) & = & (v_2 - A) + (v_1 - B) \\ \Rightarrow \frac{Ix_1}{I_s \cdot f_s(A) f_g(v_1)} \cdot \frac{Iz_{2j}}{I_s \cdot f_s(B) f_g(v_2)} & = & \frac{Ix_2}{I_s \cdot f_s(A) f_g(v_2)} \cdot \frac{Iz_{1j}}{I_s \cdot f_s(B) f_g(v_1)} \\ \Rightarrow Ix_1 \cdot Iz_{2j} & = & Ix_2 \cdot Iz_{1j}. \end{array} \tag{5.2.10}$$

This is precisely the same loop expression that we derived for the ideal loops in Section 5.1.1. We conclude that certain significant non-ideal device behaviors, including the body effect, do not detract from the accuracy of the translinear principle as applied to weak inversion MOS circuits.

5.2.4 Renormalization of current vectors.

In the idealized circuits of Section 5.2.6, only NMOS devices are used. Because PMOS and NMOS devices have symmetric behavior, any translinear circuit can be turned "upside down" and implemented with PMOS devices. To satisfy current source/load requirements, we must require that any NMOS translinear stage be followed by a PMOS stage, and any PMOS stage must be followed by an NMOS stage.

In translinear sum-product circuits, only a subset of outputs are typically desired. This is the case with the equality node circuit of Figure 5.1.7. The sum over all outputs from the Gilbert multiplier is equal to I_U, our global unit current. Because we discard some of those outputs, the output of the sum-product circuit is ultimately less than I_U. If there are many stages of sum-product circuits, this may result in steady, unwanted attenuation of currents in the circuit.

Definition 5.2.6. Current magnitude. We refer to the sum over output currents as the *current magnitude*, in units of I_U,

$$k_Z \equiv \frac{\sum_k I z_k}{I_U}. \tag{5.2.11}$$

□

To correct for the loss in current magnitude, a simple circuit such as that of Figure 5.2.6 is often used [53, 52]. The parameters n and m are geometry factors, i.e. $\left(\frac{W}{L}\right)_6 = m \cdot \left(\frac{W}{L}\right)_5$. One would typically choose $n = m = 1$. Transistor M7 is a current source whose drain current is equal to $n \cdot I_U$. M8 is usually a global device located outside of the local node. The gate voltage of M8 is distributed throughout the network as a global bias voltage.

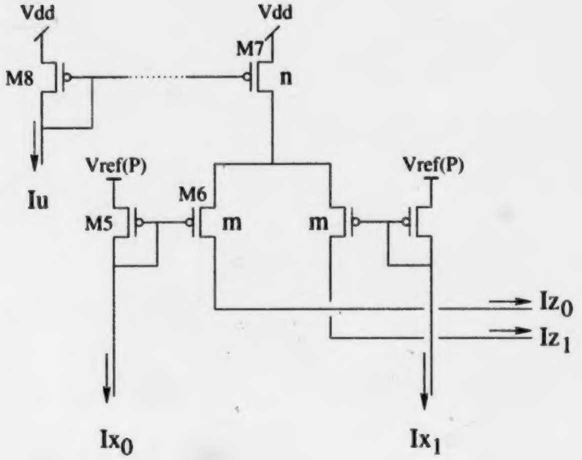


Figure 5.2.6: A PMOS renormalization circuit for a two-element current vector.

The PMOS renormalization circuit is equivalent to the translinear normalizer from Figure 5.1.5. CMOS sum-product designs are greatly simplified if all sum-product circuits are implemented in NMOS, and a PMOS normalization stage exists between all sum-product circuits. To illustrate this approach, two sum-product stages, normalizers, and their connections are shown in Figure 5.2.7. The figure demonstrates connecting an equality node to a parity-check node using CMOS translinear circuits.

5.3 * Supply voltage in canonical circuits

The requirement for saturation imposes a minimum allowed supply voltage on canonical sum-product circuits. It is common practice to use one canonical circuit made of NMOS devices, of which the outputs are "folded" into a second canonical circuit made of PMOS devices. A "slice" of this folded topology is illustrated in Figure 5.3.1.

The PMOS canonical circuit can perform either a second sum-product operation or provide renormalization of currents ("renormalization" means boosting the output currents so that their sum equals a desired unit current). The PMOS transistors must also be in saturation, and must have their own separate V_{ref}. In Figure 5.3.1,

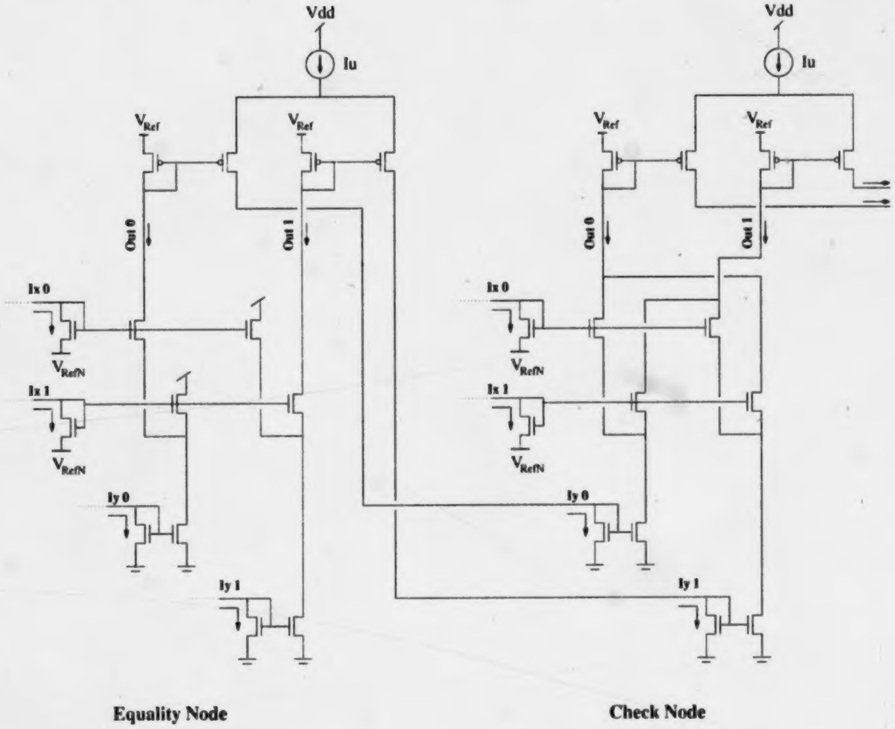

Vdd
Iu
V_Ref
V_Ref
Out 0
Out 1
Ix 0
V_RefN
Ix 1
V_RefN
Iy 0
Iy 1
Vdd
Iu
V_Ref
V_Ref
Out 0
Out 1
Ix 0
V_RefN
Ix 1
V_RefN
Iy 0
Iy 1
Equality Node
Check Node

Figure 5.2.7: Connection between two stages of NMOS sum-product circuits and PMOS normalizers.

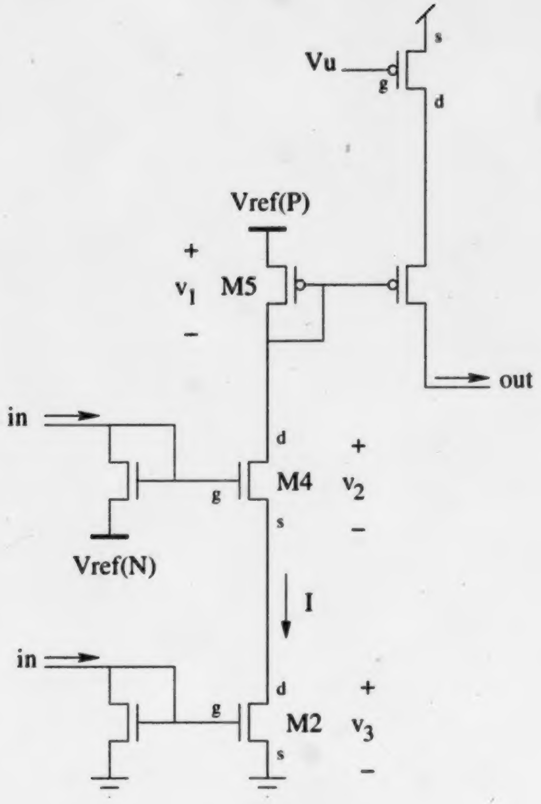


Figure 5.3.1: A "slice" of the canonical topology.

the separate V_{ref} voltages are indicated by $V_{\text{ref}}(N)$ and $V_{\text{ref}}(P)$.

5.3.1 Minimum allowable supply voltage.

It is clear in Figure 5.3.1 that the D-S terminals of M5, M4 and M2 are traversed between $V_{\text{ref}}(P)$ and ground. These three devices must remain in saturation for the circuit to function properly. Let us assume that a v_{ds} voltage of at least 200mV is required for each device

We further assume that $V_{\text{ref}}(N)$ is chosen so that $v_3 = 200mV$. To simplify the notation, we refer to V_{ref} as the "gap" between V_{dd} and $V_{\text{ref}}(P)$: $V_{\text{ref}}(P) = V_{dd} - V_{\text{ref}}$. It then follows that $v_2 = V_{dd} - V_{\text{ref}} - v_1 - 0.2V$. We also note that v_1 and v_3 are greatest when $I = I_U$, i.e. when all of circuit's current is steered through this slice.

From (5.2.6), we derive an equation for v_1:

$$\begin{aligned} v_1 &= \frac{U_T}{\kappa}\ln\left(\frac{I_U}{I_{0P}\cdot W/L}\right) && (5.3.1)\\ \Rightarrow v_2 &= V_{\mathrm{dd}} - V_{\mathrm{ref}} - 0.2V - \frac{U_T}{\kappa}\ln\left(\frac{I_U}{I_{0P}\cdot W/L}\right) \geq .2V && (5.3.2)\\ \Rightarrow V_{\mathrm{dd}} &\geq 0.4V + V_{\mathrm{ref}} + \frac{U_T}{\kappa}\ln\left(\frac{I_U}{I_{0P}\cdot W/L}\right). && (5.3.3) \end{aligned}$$

The result (5.3.3) can be arranged in a more useful form by expanding I_{0P}. It is also helpful to express I_U relative to a more realistic operating current, such as $100nA$. By expanding the division in the logarithm, and incorporating the device model (5.2.2), we arrive at

$$\begin{aligned} V_{\mathrm{dd}} \geq\ & 0.4V + V_{\mathrm{ref}} + \tfrac{U_T}{\kappa}\ln\left(\tfrac{I_U}{100nA}\right) + \tfrac{U_T}{\kappa}\ln\left(\tfrac{100nA}{1A}\right)\\ & -\tfrac{U_T}{\kappa}\ln\left(\tfrac{W}{L}\right) - \tfrac{U_T}{\kappa}\ln\left(\tfrac{2\mu_p C_{\mathrm{ox}} U_T^2}{\kappa}\right) + V_{T0P}. \end{aligned} \qquad (5.3.4)$$

This result expresses a reaonable estimation of the minimum supply voltage of canonical sum-product circuits. Numerical solutions for (5.3.4) are compared against solutions for a low-voltage topology in Section 6.2.2. Results are also plotted as functions of temperature and process feature size in Section 6.2.2.

5.3.2 Approximations.

It is possible for (5.3.4) to be simplified if certain approximations are made. First, note that $C_{\mathrm{ox}} = \frac{\varepsilon_{\mathrm{ox}}}{t_{\mathrm{ox}}}$, and consequently $\mu_p C_{\mathrm{ox}} = \varepsilon_{\mathrm{ox}}\frac{\mu_p}{t_{\mathrm{ox}}}$. If μ_p and t_{ox} are expressed in units of $\frac{\mathrm{m}^2}{V\cdot\mathrm{sec}}$ and μm, respectively, then, based on typical process values,

$$\frac{\mu_p\left(\frac{\mathrm{m}^2}{V\cdot\mathrm{sec}}\right)}{t_{\mathrm{ox}}\,(\mu\mathrm{m})} \approx 1. \qquad (5.3.5)$$

As CMOS technologies advance and the minimum feature size decreases, μ_p tends to increase while t_{ox} tends to decrease. We have evaluated the ratio (5.3.5) using parameters for several AMI and TSMC CMOS processes, ranging in feature size from $1.5\mu m$ to $0.18\mu m$. The value of the mobility-to-oxide thickness ratio (5.3.5) was found to be within the range of 0.6 to 2.0 for all processes. The data for these processes was obtained through MOSIS.

The ratio (5.3.5) therefore tends to increase in more advanced processes. Recalling that $\frac{U_T}{\kappa} \approx 0.0357$, the corresponding logarithmic term, $\frac{U_T}{\kappa} \ln\left(\frac{\mu_p}{t_{\mathrm{ox}}(\mu\mathrm{m})}\right)$, is appreciable only when (5.3.5) is increased by almost two orders of magnitude. In TSMC and AMI processes, the logarithmic term is at most $0.035V$, and more typically $0.013V$. The logarithmic term is always positive in sub-micron processes, and therefore can only improve the bound on V_{dd}. It is therefore reasonable to neglect the effect of (5.3.5) in (5.3.4). We then arrive at

$$\begin{aligned} V_{\mathrm{dd}} \quad \geq \quad & 0.4V + V_{\mathrm{ref}} + \tfrac{U_T}{\kappa} \ln\left(\tfrac{I_U}{100nA}\right) + \tfrac{U_T}{\kappa} \ln\left(\tfrac{100nA}{1A}\right) \\ & - \tfrac{U_T}{\kappa} \ln\left(\tfrac{W}{L}\right) - \tfrac{U_T}{\kappa} \ln\left(\tfrac{2\varepsilon_{\mathrm{ox}} U_T^2}{\kappa} \cdot \tfrac{10^6 \mu\mathrm{m}}{1\mathrm{m}}\right) + V_{TOP}. \end{aligned} \tag{5.3.6}$$

Finally, we may substitute approximate values for U_T and κ, and neglect the relatively small W/L term, arriving at the final bound

$$V_{\mathrm{dd}} \geq 0.42V + V_{\mathrm{ref}} + V_{TOP} + \frac{U_T}{\kappa} \ln\left(\frac{I_U}{100nA}\right). \tag{5.3.7}$$

As a concrete example, a typical 0.18μm CMOS logic process has $V_{TOP} = 0.39V$. For a canonical analog sum-product decoder implemented in this process, with a $100nA$ operating current, (5.3.7) requires $V_{\mathrm{dd}} > 0.81 + V_{\mathrm{ref}}$. At best, $V_{\mathrm{ref}} > 0.2V$, therefore $V_{\mathrm{dd}} > 1V$ is required.

5.4 *A reduced-complexity sum-product circuit.

The canonical CMOS sum-product topology of Figure 5.2.4 can sometimes be modified to significantly reduce complexity. Consider, for example, the case in which the inputs X and Y each have degree N, and we desire only the N products $x_i \cdot y_i$. The canonical circuit produces N^2 products, wasting $N^2 - N$ transistors.

This is necessary in the canonical circuit because all X inputs must be present in each column, so that $\sum_i Ix_i = I_U$ is in the denominator. If only a subset $\delta X \subset X$ is used as input, then the denominator of (5.1.11) is not a constant (in probability terms).

To solve this problem, we introduce a *reference* input Δ, so that $\sum_{i \in \delta X} Ix_i + \Delta = I_U$. This approach is illustrated in Figure 5.4.1. In this example, $\{a, b, c, d\}$ is a subset of the alphabet $\mathcal{A}_x$, so that $\delta X = \{Ix_a, Ix_b, Ix_c, Ix_d\}$, and $\sum_{i \in \delta X} Ix_i \leq I_U$.

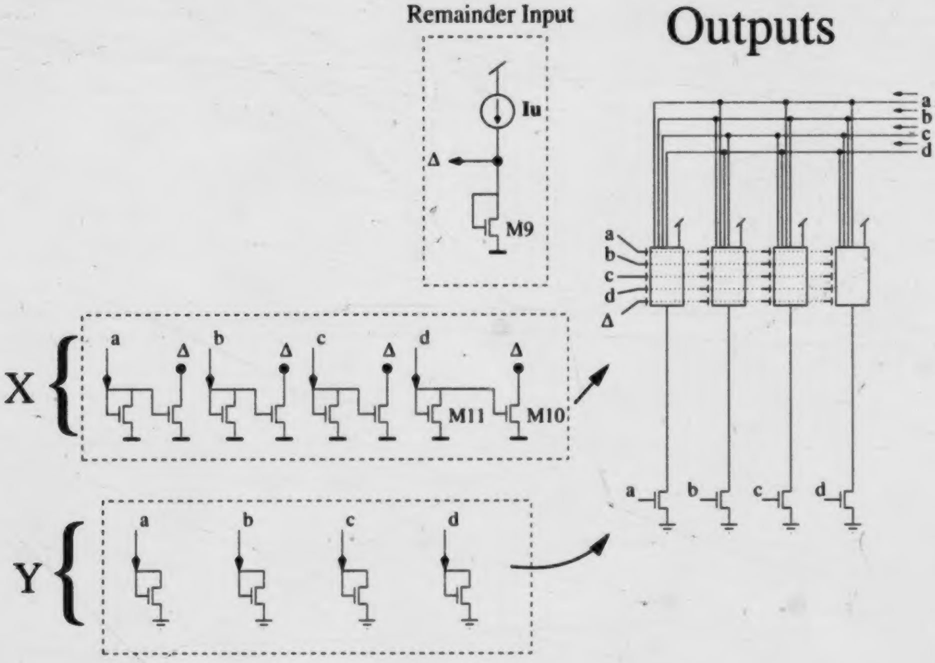


Figure 5.4.1: Use of a reference input to restore the denominator of (5.1.12).

The reference input is created by transistor M9 at node Δ. The other inputs arrive normally, as at M11. At each input, a second transistor (e.g. M10) acts as a current mirror to replicate the input. All such replicated input currents are connected to node Δ.

A current I_U is supplied to node Δ, and then a current equal to $\sum_{i \in \delta X} Ix_i$ is siphoned away by the replicated input currents. At each source-connected box, the input nodes are δX and Δ. The denominator of the outputs is therefore equal to $\sum_{i \in \delta X} Ix_i + \Delta = I_U$. The denominator is therefore restored and can be neglected as usual.

5.4.1 *Complexity of the reference circuit.

Let $\delta N = |\delta X|$ and $\delta M = |\delta Y|$, and let $N = |X|$ and $M = |Y|$. We wish to compute all pairs of products for inputs in the sets δX and δY. In the canonical circuit, we would have $N + 2M + MN$ transistors. In the reference circuit, we have $2(\delta N + \delta M) +$

$\delta M(\delta N + 1) + 2$ transistors.

This analysis is overly simplistic, because it may be necessary to compute products for several subsets. There are many possible scenarios with different complexity results.

Example 5.4.1. Diagonal products. Suppose $M = N$ and we only want the diagonal products $Ix_i \cdot Iy_i$ for $i = 1, ..., N$. We construct a separate reference circuit for each of these products, so that $\delta N = \delta M = 1$. There are then eight transistors per product, and N such products, leading to a total of $8N$ transistors. The canonical circuit would require $3N + N^2$ transistors. The reference circuit for this situation is more efficient when $N \geq 5$.

□

Example 5.4.2. Disjoint subsets. A set of products is divided into disjoint subsets $\delta X_1 \otimes \delta Y_1$ and $\delta X_2 \otimes \delta Y_2$ (where $X \otimes Y$ denotes the set of all pairwise products xy for $x \in X$ and $y \in Y$). This is represented by the trellis section in Figure 5.4.2. This section is taken from the conventional $(16, 11)$ Hamming code trellis, produced by the squaring construction in Example 3.3.4.

We wish to construct a sum-product circuit to calculate the probabilities of Z given those of X and Y. In the canonical approach, we would need $N \cdot (3 + N)$ transistors, with $N = 8$, thus requiring 88 transistors. This approach would also produce 32 unused outputs.

If we use reference inputs, then two subset products are required. Each subset needs $2(8) + 4(5) + 2 = 38$ transistors, for a total of 76. There is a net savings of twelve transistors with the reference approach.

□

A more important advantage of the reference circuit in Example 5.4.2 is the simplicity of design. The disjoint trellis subsections are isomorphic to each other. We only need to design one simple circuit to implement the sum-product algorithm for a subsection. The same circuit can be reused for the other subsection.

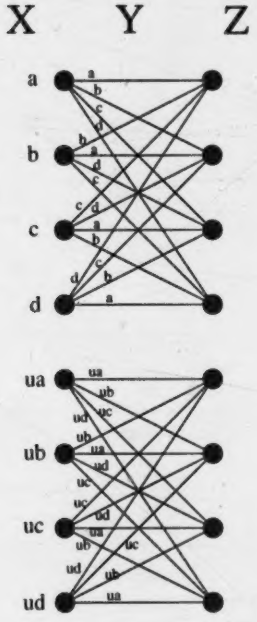


Figure 5.4.2: A trellis section with disjoint subsections.

When a trellis is constructed of disjoint subtrellises, as is the case with the squaring construction, the reference circuit allows us to implement sum-product circuits for the subtrellises only. These subtrellis implementations are then duplicated to produce the complete decoder.

5.4.2 *Implementing all directions.

Full implementation of a three-edge sum-product node requires three directions of computation, which we call forward, backward, and upward. When implementing a trellis using disjoint subtrellises, as in Example 5.4.2, it is not always necessary to use the reference circuit.

A reference input is required only when the circuit's *row inputs* are split up among the subtrellises. If one of the edges carries a binary variable, then the computations can be arranged so that the binary variable is the row input. Disjoint circuits can then be used without requiring a reference input.

This scenario is illustrated by the trellis section in Figure 5.4.3, in which the

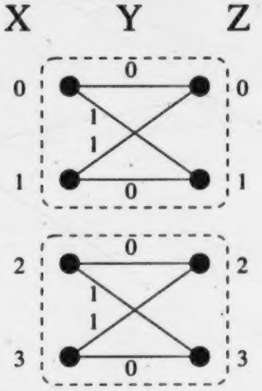


Figure 5.4.3: A trellis section with disjoint "butterfly" subtrellises.

variables X and Z are quaternary, while Y is binary. The trellis consists of two "butterfly" structures, which can be implemented using a single simple circuit. Implementation of this trellis section is illustrated in Figure 5.4.4, in which dashed lines represent incomplete probability masses, solid lines represent complete probability masses, and a solid dot represents the edge designated as the row input.

In the forward direction, where output is on edge Z, the X inputs may be split among the column inputs of two butterfly circuits, while Y arrives as the row input. A similar arrangement is possible in the backward direction. In these cases, the row input is always a complete probability mass, so no reference input is needed.

In the upward computation, with output on the Y edge, either X or Z must be chosen as the row input. Splitting the row input's probability mass is unavoidable in this case, so a reference input is necessary.

To fully implement this trellis section with sum-product circuits, only two basic circuits are needed. These are the butterfly circuit with no reference input, and the butterfly circuit with a reference input. With no reference input, the butterfly circuit is the same as the "check node" circuit on the right-hand side of Figure 5.2.7. A butterfly circuit with reference input is shown in Figure 5.4.5.

The reference circuit allows us to reduce complex sum-product circuits to simple primitive subtrellises. To create a complete decoder, it therefore suffices to have only a small library of subtrellis circuits, which also facilitates synthesis of analog

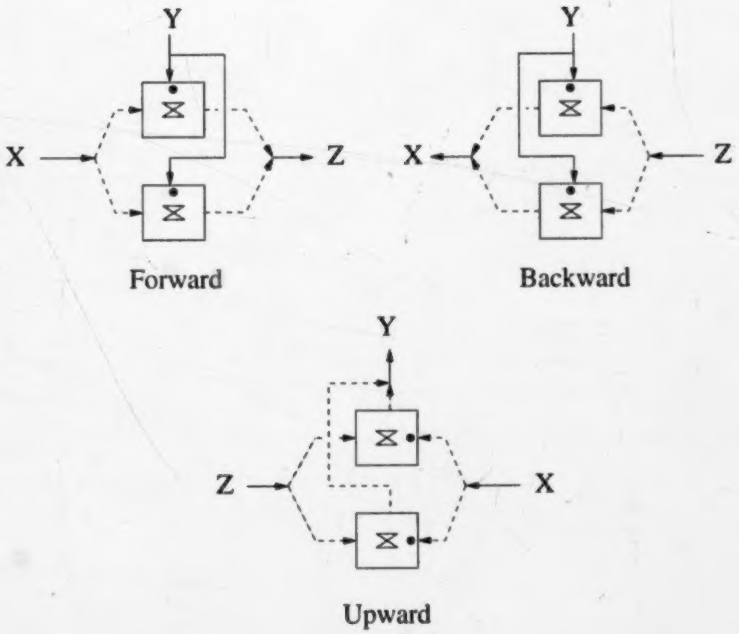

Y
X
Z
Forward
Y
X
Z
Backward
Y
Z
X
Upward

Figure 5.4.4: Implementation of a trellis section with disjoint butterfly subtrellises. Dashed lines indicate incomplete probability masses. Solid lines indicate complete probability masses. A dot indicates the row input edge for each cell.

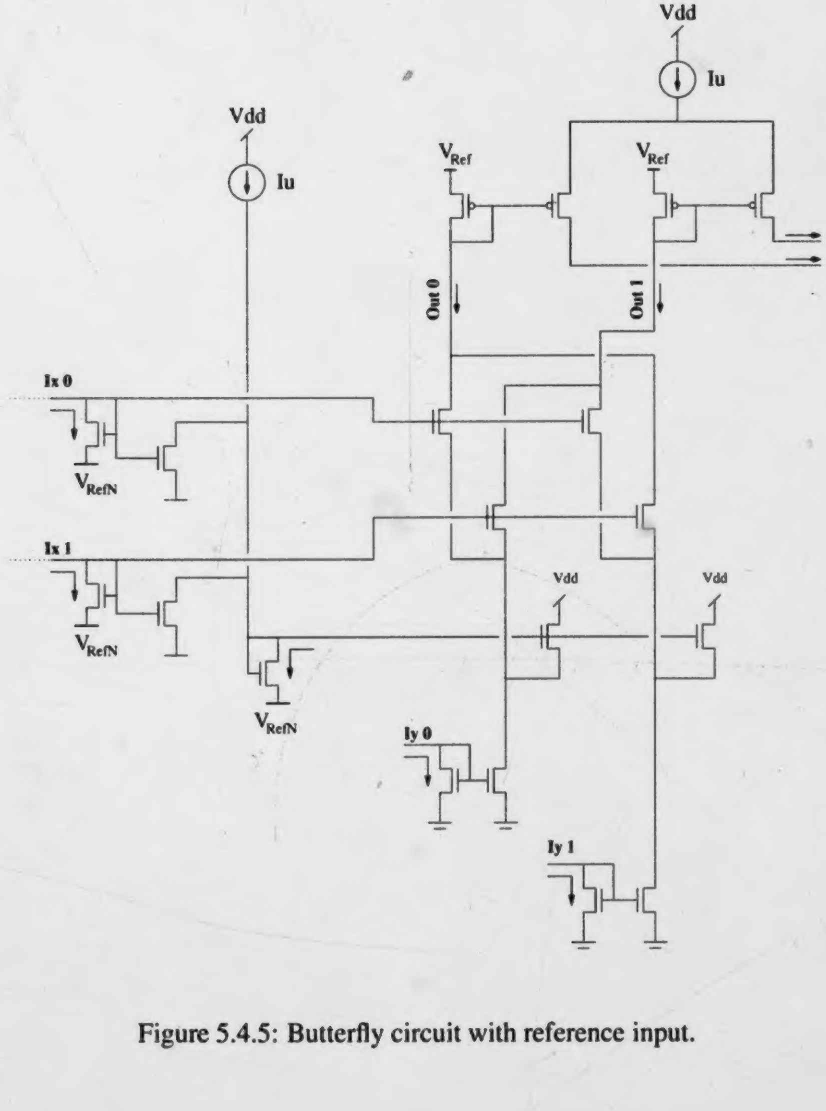

Vdd
Iu
Vdd
Iu
V_Ref
V_Ref
Out 0
Out 1
Ix 0
V_RefN
Ix 1
V_RefN
Vdd
Vdd
V_RefN
Iy 0
Iy 1

Figure 5.4.5: Butterfly circuit with reference input.

sum-product circuits. Approaches to synthesis based on subcircuit libraries, including circuits with reference inputs, were explored by the author of this thesis [25], and in the doctoral thesis of Jie Dai [24].

Chapter 6

*Low-voltage Analog Decoding Circuits

6.1 Eliminating V_{ref}

It is clear from Section 5.3, in particular from (5.3.7), that the voltage needs of the canonical circuit can be reduced if we allow $V_{\text{ref}} = 0$ in (5.3.7). If $V_{\text{ref}} = 0$, M2 and related transistors (see Figures 5.2.4 and 5.3.1) are not in saturation. When this happens, the reverse current term I_r becomes significant in (5.2.7).

This situation can be analyzed using the translinear principle, in which we regard I_r as a second, parallel translinear current controlled by v_{ds}. We apply the model introduced by Definition 5.2.4, as illustrated in Figure 5.2.3. Our analysis in this section is based on the method introduced in [74].

With M2 unsaturated, the circuit consists of the translinear loops shown in Figures 6.1.1, 6.1.2, and 6.1.3. The first two loops were studied in Section 5.2.3, and yield the equations

$$
\begin{aligned}
Ix_j \cdot Iz_{ik} &= Iz_{ij} \cdot Ix_k && (6.1.1)\\
If &= Iy_i. && (6.1.2)
\end{aligned}
$$

Figure 6.1.2 introduces the unsaturated transistor, with its additional component Ir. Also introduced is the current Id_i, defined as

$$
Id_i \equiv If - Ir. \qquad (6.1.3)
$$

We also recognize the role of the source-connected transistors, as in (5.1.10), so

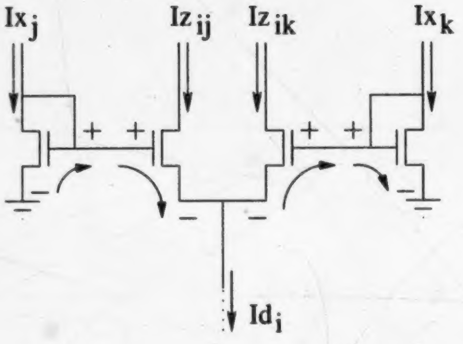


Figure 6.1.1: Translinear loop 1, derived from Fig. 5.2.4 with $V_{ref} = 0$.

that

$$Id_i = \sum_k Iz_{ki}. \tag{6.1.4}$$

To solve for all currents in the circuit, we need one more equation, which is provided by the third loop shown in Figure 6.1.3:

$$\begin{aligned} Iz_{ij} \cdot If &= Ir \cdot Ix_j && (6.1.5)\\ \Rightarrow Ir &= Iy_i \cdot \frac{Iz_{ij}}{Ix_j}. && (6.1.6) \end{aligned}$$

Combining (6.1.1) through (6.1.6), we arrive at

$$\begin{aligned} Id_i &= Iy_i - Iy_i \cdot \frac{Iz_{ij}}{Ix_j} && (6.1.7)\\ \Rightarrow \sum_k Iz_{ki} &= Iy_i - Iy_i \cdot \frac{Iz_{ij}}{Ix_j} && (6.1.8)\\ \Rightarrow \frac{Iz_{ij}}{Ix_j} \sum_k Ix_k + Iy_i \cdot \frac{Iz_{ij}}{Ix_j} &= Iy_i && (6.1.9)\\ \Rightarrow Iz_{ij} &= \frac{Iy_i \cdot Ix_j}{\sum_k Ix_k + Iy_i}. && (6.1.10) \end{aligned}$$

The result (6.1.10) is almost the same as the normal canonical output (5.1.11), except there is an additional term in the denominator. In probability terms, the denominator of (6.1.10) can no longer be neglected.

6.2 Low-voltage sum-product circuits

To solve the problem posed by the denominator of (6.1.10), additional *dummy transistors* may be added with their sources connected to Id_i. If these transistors repre-

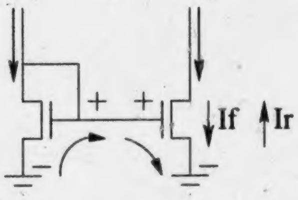

Iy_i
Id_i
+
+
−
−
If
Ir

Figure 6.1.2: Translinear loop 2, derived from Fig. 5.2.4.

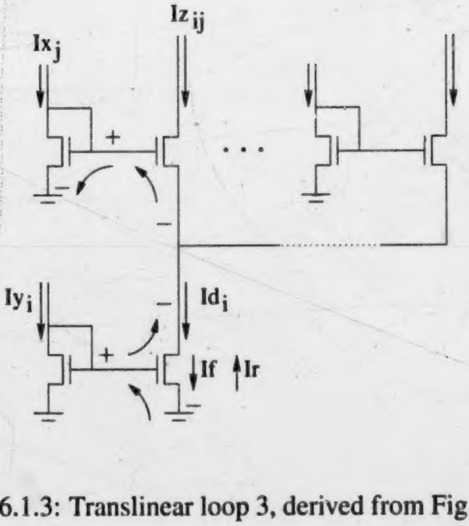

Iz_ij
Ix_j
+
−
−
. . .
Iy_i
Id_i
−
+
If
Ir

Figure 6.1.3: Translinear loop 3, derived from Fig. 5.2.4.

sent a current $I_\delta \equiv \sum_{l \neq i} Iy_l$, then the output becomes

$$\begin{aligned} Iz_{ij} &= \frac{Iy_i \cdot Ix_j}{\sum_k Ix_k + I_\delta + Iy_i} && (6.2.1) \\ \Rightarrow \; Iz_{ij} &= \frac{Iy_i \cdot Ix_j}{\sum_k Ix_k + \sum_{l \neq i} Iy_l + Iy_i} && (6.2.2) \\ \Rightarrow \; Iz_{ij} &= \frac{Iy_i \cdot Ix_j}{\sum_k Ix_k + \sum_l Iy_l}. && (6.2.3) \end{aligned}$$

In probability terms, the denominator of (6.2.3) is a constant and can be neglected. The addition of redundant transistors therefore corrects the probability calculation of the canonical topology when $V_{\text{ref}} = 0$. Because the outputs of these new transistors are of no use, we refer to them as *dummy transistors* or dummy inputs. The drains of these transistors are simply connected to V_{dd}.

6.2.1 A general low-voltage sum-product topology

Figure 6.2.1 displays a general low-voltage sum-product circuit topology based on these results. In Figure 6.2.1, X denotes the ordered sequence of row inputs (voltages) $< x_1, ..., x_N >$, and Y_k denotes the set of column inputs (voltages) $\{y_l : 1 \leq l \leq M \text{ and } l \neq k\}$ which *excludes* the input y_k. The members of Y_k need have no particular order.

The current inputs Ix_i and Iy_k are converted into voltage inputs x_i and y_k by the diode-connected transistors on the left. Apart from the PMOS transistors used in the Renormalization circuit, all devices have the same dimensions. This circuit produces all pairs of products of inputs from X and Y. To complete the sum-product computation (Section 2.4.2), desired products are summed together by shorting wires (indicated by the "Connectivity" block). Unused products must be shorted to V_{dd}.

6.2.2 Supply voltage in low-voltage circuits

To calculate the minimum allowed supply voltage in the low-voltage topology, we repeat the analysis of Section 5.3 with appropriate modifications. Figure 6.2.2 shows a slice of the low-voltage topology, which is nearly the same as the canonical slice of Figure 5.3.1.

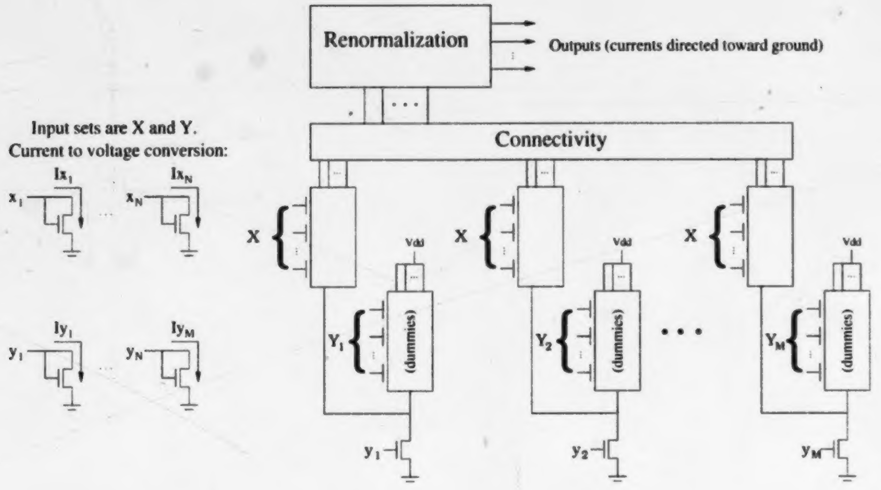

Renormalization
Outputs (currents directed toward ground)
Input sets are X and Y.
Current to voltage conversion:
Ix_1
Ix_N
x_1
x_N
Iy_1
Iy_M
y_1
y_N
Connectivity
X
X
X
Vdd
Vdd
Vdd
Y_1
Y_2
Y_M
(dummies)
(dummies)
(dummies)
y_1
y_2
y_M

Figure 6.2.1: Low-voltage sum-product circuit topology, using the box notation of Fig. 5.2.5.

We again assume that all current is steered through this slice. In the low-voltage topology, we may interpret I_U as a maximum current level in the circuit (in Section 6.2.5 we discuss how the operating current is somewhat variable in low-voltage circuits). To address the worst-case situation, we assume that both inputs in Figure 6.2.2 are equal to I_U. It can be easily verified in this case that $I = 0.5 \cdot I_U$.

We no longer require M2 to be in saturation, but for proper function it is still necessary to have M4 in saturation. Again applying the device model (5.2.6), we arrive at the equations

$$\begin{aligned} v_1 &= \frac{U_T}{\kappa} \ln\left(\frac{0.5 \cdot I_U}{I_{0P}}\right) \\ v_2 &= V_{\text{dd}} - \frac{U_T}{\kappa} \ln\left(\frac{0.5 \cdot I_U}{I_{0P}}\right) - v_3. \end{aligned} \tag{6.2.4}$$

To solve for v_3, we assume that M2 is in saturation, that M4 is not in saturation, that both devices share the same gate voltage, v_g, and that both devices share the same device current. Then

$$\begin{aligned} I_{0N} \frac{W}{L} e^{\frac{\kappa v_g}{U_T}} \left[1 - e^{-\frac{v_3}{U_T}}\right] &= I_{0N} \frac{W}{L} e^{\frac{\kappa v_g}{U_T}} \left[e^{-\frac{v_3}{U_T}}\right] \\ \Rightarrow 1 - e^{-\frac{v_3}{U_T}} &= e^{-\frac{v_3}{U_T}} \\ \Rightarrow v_3 &= U_T \ln(0.5). \end{aligned} \tag{6.2.5}$$

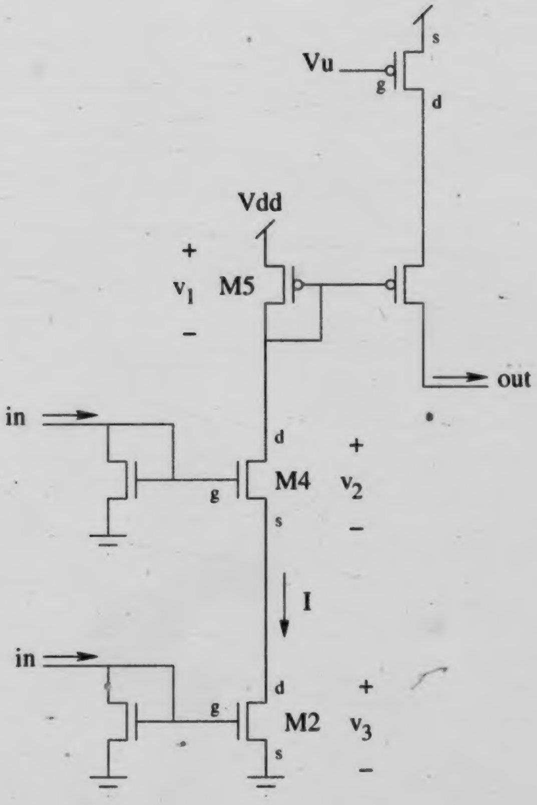

Vu
g
s
d
Vdd
+
v1
M5
−
out
in
d
g
M4
s
+
v2
−
I
in
d
g
M2
s
+
v3
−

Figure 6.2.2: A "slice" of the low-voltage topology.

We now substitute (6.2.5) into (6.2.4) to obtain

$$\begin{aligned} v_2 &= V_{dd} - \frac{U_T}{\kappa}\ln\left(\frac{0.5\cdot I_U}{I_{0P}}\right) - U_T\ln(0.5) \geq .2V \\ \Rightarrow V_{dd} &\geq 0.2V + \frac{U_T}{\kappa}\ln\left(\frac{0.5\cdot I_U}{I_{0P}}\right) + U_T\ln(0.5). \end{aligned} \tag{6.2.6}$$

6.2.3 Approximations.

It is again possible to express this bound in a more usable form. We expand I_{0P}, making the same approximations and adjusting the units as in Section 5.3. After organizing all such adjustments and substituting approximations, we arrive at the final form:

$$V_{dd} \geq 0.211 + V_{T0P} + \frac{U_T}{\kappa}\ln\left(\frac{I_U}{100nA}\right). \tag{6.2.7}$$

Comparing (6.2.7) with (5.3.7), we find that the low-voltage topology allows V_{dd} to be reduced by at least $0.4V$ (assuming $V_{ref} > 0.2V$). For the typical 0.18μm process referenced in Section 5.3, $V_{T0P} = 0.39$. For this process, a low-voltage decoder is constrained by $V_{dd} > 0.61V$.

6.2.4 Process scaling and temperature effects.

A more accurate estimate of minimum supply voltage can be made by accounting explicitly for all parameters, and solving numerically. The mobility parameter, μ, the thermal voltage, U_T, and the threshold voltage, V_{T0P}, all depend on temperature. These parameters also depend on a host of other process conditions such as doping concentration and oxide thickness.

A collection of semi-empirical models are suggested by Mead in [51]. These models relate oxide thickness, impurity doping, threshold voltage, and other process parameters to the scaling of minimum feature-size in sub-micron processes. Using these models, together with models for parametric dependence on temperature, it is possible to evaluate the actual minimum supply voltage of canonical and low-voltage circuits, as a function of temperature and process scaling.

First, the dependence of mobility on temperature is given by [63]

$$\mu = 54.3 \cdot T_n^{-0.57} + \frac{7.4 \cdot 10^8 \cdot T^{-2.23}}{1 + \frac{N}{2.35 \cdot 10^{17} \cdot T_n^{2.4}} \cdot 0.88 \cdot T_n^{-0.146}}, \tag{6.2.8}$$

where T is the temperature in K, $T_n = T/300$, and N is the total dopant concentration.

According to Mead, the dopant density is approximately

$$N \approx 4 \times 10^{16} L_{\min}^{-1.6} \tag{6.2.9}$$

where $L_{\min}$ is the minimum feature size of the process, specified in units of μm. Similarly, the oxide thickness is estimated by

$$t_{\mathrm{ox}} \approx \max\left(210 L_{\min}^{0.77}, 140 L_{\min}^{0.55}\right). \tag{6.2.10}$$

The nominal threshold voltage scales as

$$V_{T0} \approx 0.55 L_{\min}^{0.23}. \tag{6.2.11}$$

The relationship between V_T and temperature is approximately expressed by

$$V_T(T) \approx V_{T0} - 2\psi_{B0} + 2\psi_B(T), \tag{6.2.12}$$

where ψ_{B0} is the value of ψ_B at room temperature, and ψ_B is given by

$$\psi_B(T) = U_T \ln\left(\frac{N}{n_i}\right), \tag{6.2.13}$$

and where n_i is the density of carriers in intrinsic (undoped) silicon ($n_i = 1.45 \times 10^{10} \mathrm{cm}^{-3}$).

The subthreshold slope factor, κ, is a function of ψ_B, N, C_{ox}, and the gate-to-bulk voltage. We approximate κ by its near-minimum value, which occurs near the boundary between weak-inversion and depletion. We begin with the depletion depth under the gate, which is [51]

$$d = \sqrt{\frac{2\varepsilon_s \psi_B}{qN}}, \tag{6.2.14}$$

where ε_s is the permittivity of silicon, and q is the charge on the electron. The corresponding depletion-layer capacitance is then given by

$$C_d = \frac{\varepsilon_s}{d}. \tag{6.2.15}$$

The oxide capacitance, C_{ox}, is

$$C_{\mathrm{ox}} = \frac{\varepsilon_{\mathrm{ox}}}{t_{\mathrm{ox}}}, \tag{6.2.16}$$

and κ is defined as the capacitive divider relation between C_{ox} and C_d:

$$\kappa \approx \frac{C_{\mathrm{ox}}}{C_{\mathrm{ox}} + C_d}. \tag{6.2.17}$$

These equations are collected and solved numerically, yielding estimates of (6.2.6) and (5.3.4) as functions of temperature and feature size. The results of this calculation are reported in Figure 6.2.3. The difference in allowable supply voltage between canonical and low-voltage circuits is shown in Figure 6.2.4 as a function of temperature. The supply-voltage reduction achieved by low-voltage circuits depends only weakly on temperature, and evidently improves as the temperature is increased.

6.2.5 Renormalization

For successful implementation of a low-voltage decoder, renormalization of currents between modules is essential. In a canonical sum-product circuit described by (5.1.11), the denominator is equivalent to a probability of one and can be truly ignored. In a low voltage circuit described by (6.2.3), however, the denominator is equivalent to two, substantially reducing the current magnitude at the output of each module.

In principle, linear attenuation will not change the result of decoding. The sum-product algorithm only depends on the relative proportions among input currents (relative to each other), not on their precise magnitude. Repeated attenuation in a large network, however, will cause the outputs to approach zero, making it impossible to extract any results.

In a practical setting, the sum-product algorithm is carried out repeatedly in a large network of sum-product nodes. The output of one node provides input for the

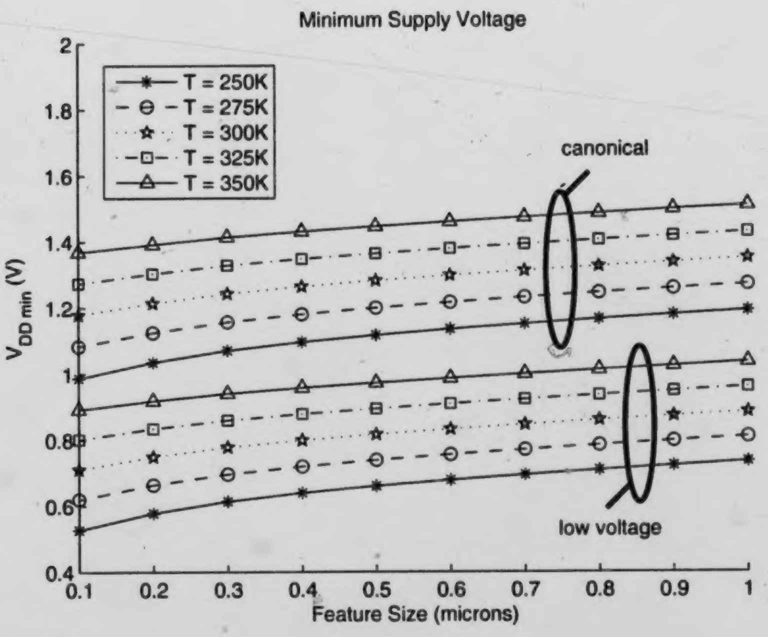


Figure 6.2.3: Allowable supply voltages for canonical and low-voltage topologies, as a function of process feature size and temperature.

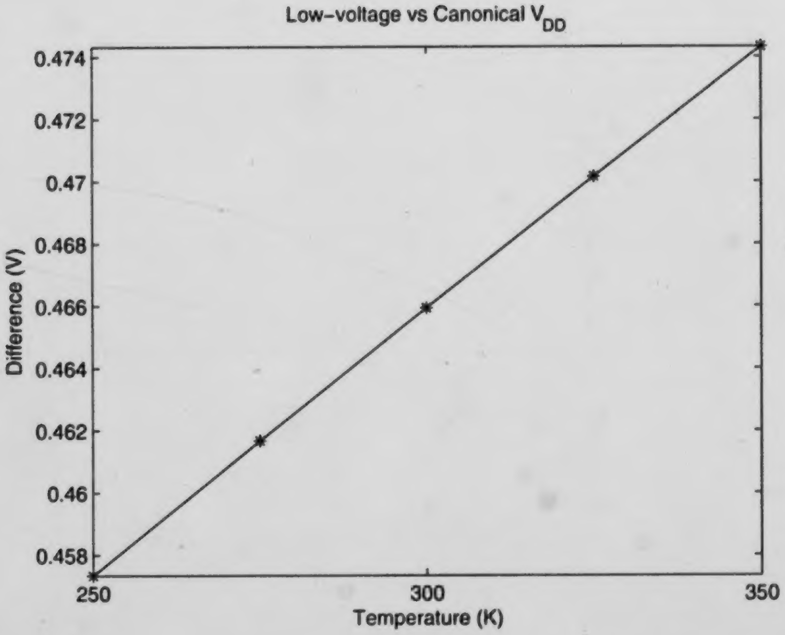


Figure 6.2.4: Difference in minimum supply voltage between low-voltage and canonical sum-product circuits, as a function of temperature.

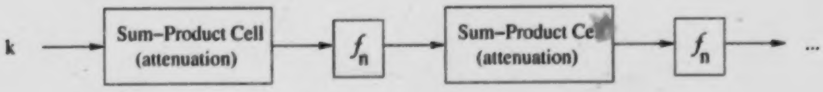

k
Sum–Product Cell (attenuation)
f_n
Sum–Product Cell (attenuation)
f_n
...

Figure 6.2.5: Iterated amplification of current magnitude *k*.

next. By "unwrapping" the decoding network, the decoder may be regarded as a cascaded array of sum-product nodes. In terms of the current magnitude (Definition 5.2.6), the circuit becomes an iteration of attenuation and renormalization.

By inserting a suitably designed renormalization circuit between modules, the currents are prevented from approaching zero. We utilize the circuit of Figure 5.2.6 with $V_{\text{ref}}(P) = V_{\text{dd}}$. The circuit is now a form of the low-voltage topology described by (6.2.3), so that the output is

$$Iz_i = \frac{n \cdot m \cdot I_u \cdot Ix_i}{n \cdot I_u + m \cdot \sum_k Ix_k}. \tag{6.2.18}$$

As in (5.1.7), the low-voltage renormalization behavior (6.2.18) amplifies each input by a constant factor. The output current magnitude, however, is not exactly I_U, and may in fact be significantly less than I_U.

It is possible to choose n and m in Figure 5.2.6 so that *upon iteration* the current magnitude, k, converges to a controlled fixed point greater than zero. In a parallel analog decoder, these iterations occur over *space*, through successive nodes within the network. The iterative process is illustrated in Figure 6.2.5, where f_n is the transfer function of the low-voltage renormalization circuit.

This allows us to treat (6.2.18) as a simple one-dimensional transfer function,

$$k_{Out} = f_n(k_{in}) = \frac{n \cdot m \cdot k_{In}}{n + m \cdot k_{In}}. \tag{6.2.19}$$

To determine the dynamic behavior of this system, we identify the *fixed points* (where $k_{Out} = k_{In}$) and determine whether they are *stable*. It is easy to verify that the fixed points occur at

$$k_0 = 0 \tag{6.2.20}$$

$$k_1 = n - \frac{n}{m}. \tag{6.2.21}$$

It is well known that a fixed point is stable and non-oscillating if and only if the slope of the transfer function, $f_n'(k)$, satisfies $0 \leq f_n'(k_f) \leq 1$ at the fixed point k_f. Also, a fixed point k_f is unstable (i.e. it is a repeller) if and only if $f_n'(k_f) > 1$. It is again easy to verify that

$$f_n'(0) = m \tag{6.2.22}$$

$$f_n'(k_1) = \frac{1}{m}. \tag{6.2.23}$$

(6.2.22) and (6.2.23) show that there is always a stable fixed point above zero when $m > 1$. The canonical renormalizer uses $m = 1$, in which case there is no fixed point greater than zero, thus driving all currents to zero in a low-voltage network. By simply using $m > 1$ this can be avoided.

The transfer function (6.2.19) is shown for various values of m in Figure 6.2.6, in which $n = 1.2$. The iterated behavior is also illustrated, with fixed points represented by large circles. Figure 6.2.6 demonstrates that, with sufficiently large m, the normalized current magnitude can be driven very close to the desired operating point with only one iteration.

6.3 Decoder architectures

6.3.1 Trellis decoders

One very common class of decoders employ the BCJR algorithm, which is described in Section 3.2.3. Trellis-based MAP decoders are used to construct serial and parallel concatenated Turbo codes.

An example of the low-voltage sum-product architecture for computing on trellis graphs is shown in Figure 6.3.1. The branch variable y takes values from the set $\{a, b, c, d\}$. The sum-product equation for this particular trellis section can be written in matrix form as [7]

$$\begin{bmatrix} \mathrm{Out0} \\ \mathrm{Out1} \\ \mathrm{Out2} \\ \mathrm{Out3} \end{bmatrix} = \begin{bmatrix} a & & d & \\ b & & c & \\ & c & & b \\ & d & & a \end{bmatrix} \begin{bmatrix} \alpha 0 \\ \alpha 1 \\ \alpha 2 \\ \alpha 3 \end{bmatrix}. \tag{6.3.1}$$

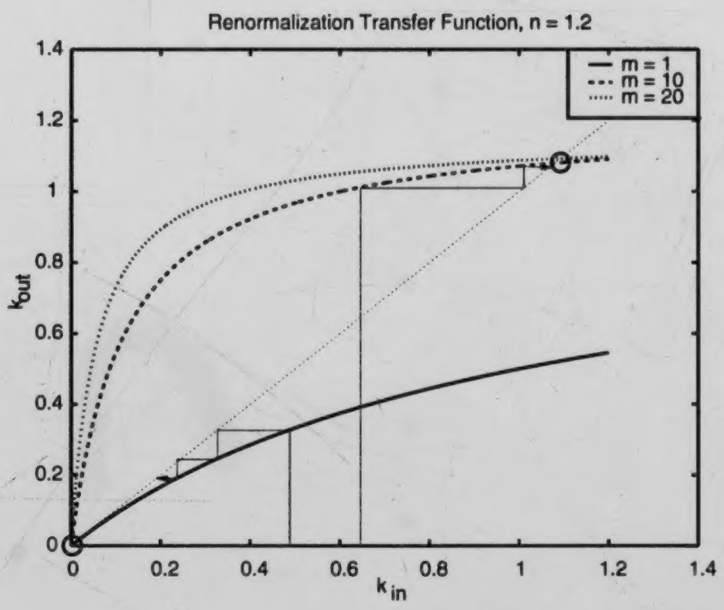


Figure 6.2.6: Current magnitude transfer function for the low-voltage renormalization circuit.

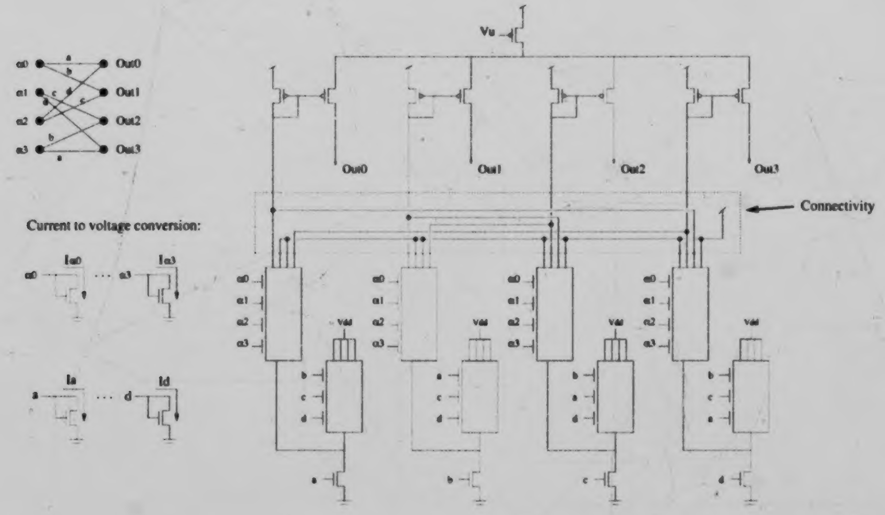

α0
α1
α2
α3
Out0
Out1
Out2
Out3
a
b
c
d
Current to voltage conversion:
Iα0
Iα3
Ia
Id
Vu
Vdd
Connectivity

Figure 6.3.1: A low-voltage circuit for trellis decoding based on the BCJR algorithm.

The particular trellis function is determined by the Connectivity section. Also shown is the Renormalization circuit. The circuit is shown with the state probabilities as row inputs and the branch probabilities as column inputs, but these roles can be reversed without affecting the results. Every stage of the BCJR algorithm consists of a matrix multiplication of the form (6.3.1). Low-voltage sum-product circuits can therefore be produced to implement a complete BCJR decoder of arbitrary size.

6.3.2 LDPC (Tanner Graph) Decoders

Decoders for binary LDPC codes are mapped from the code's "normalized" Tanner Graph, which is a direct visualization of the code's binary parity check matrix, as explained in Section 2.3.2. The Tanner Graph contains two types of constraint nodes: check nodes and equality nodes. All variables in the graph are binary. For implementation, this means that all probability masses have only two components.

For a three-edge check node, the constraint function is simply a logical XOR operation. Applying this to the sum-product algorithm (Section 2.4.2), and labeling the three edges X, Y, and Z, we obtain

$$\begin{aligned} P(Z=0) &= P(X=0)\cdot P(Y=0)+P(X=1)\cdot P(Y=1) && (6.3.2)\\ P(Z=1) &= P(X=1)\cdot P(Y=1)+P(X=0)\cdot P(Y=1). && (6.3.3) \end{aligned}$$

For a three-edge equality node, we have

$$\begin{aligned} P(Z=0) &\propto P(X=0)\cdot P(Y=0) && (6.3.4)\\ P(Z=1) &\propto P(X=1)\cdot P(Y=1). && (6.3.5) \end{aligned}$$

Applying the general circuit of Figure 6.2.1, and varying the connectivity to produce the appropriate functions, we arrive at the circuits of Figures 6.3.2 and 6.3.3. The 'y' inputs are the column inputs, and the 'x' inputs are the row inputs. M3 and M4 are dummies. Complete decoders for linear binary block codes, including LDPC codes, can be constructed from these three-edge circuits.

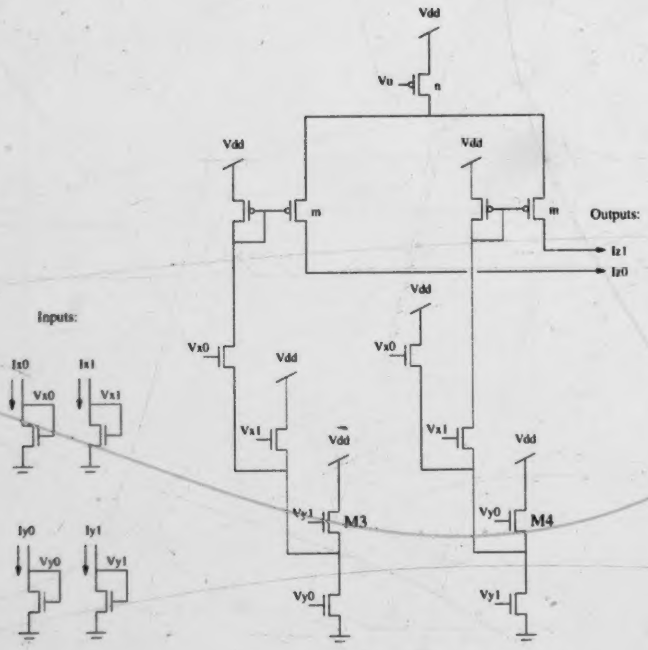

Vdd
Vu
n
Vdd
Vdd
m
m
Outputs:
Iz1
Iz0
Vdd
Inputs:
Ix0
Ix1
Vx0
Vx1
Vx0
Vdd
Vx0
Vx1
Vx1
Vdd
Vdd
Vy1
M3
Vy0
M4
Iy0
Iy1
Vy0
Vy1
Vy0
Vy1

Figure 6.3.2: Low-voltage equality-node circuit for LDPC decoding.

6.4 An example decoder

A low-voltage LDPC-style decoder for a (7,4) Hamming code is shown in Figure 6.4.1. The circles labeled x_i represent the channel sample probabilities, which are the decoder's input. The outputs emerge from the top-level equality nodes. The triangles represent comparators, which make hard decisions based on the output probability currents.

The boxes labeled '=' and '+' represent the circuits of Figures 6.3.2 and 6.3.3, respectively. A separate sum-product circuit is required for each of the node's edges, as indicated in Figure 6.4.2. In this design, the base MOS transistor size is $L = 0.18\mu$m and $W = 2\mu$m. The renormalization parameters are $M = 20$ and $N = 4$. The decoder is simulated using models for a TSMC 0.18μm digital CMOS technology.

Simulation results for this decoder, operating at a supply voltage of $0.6V$, are presented in Figure 6.4.3. These results were obtained using BSIM3 V3.1 models,

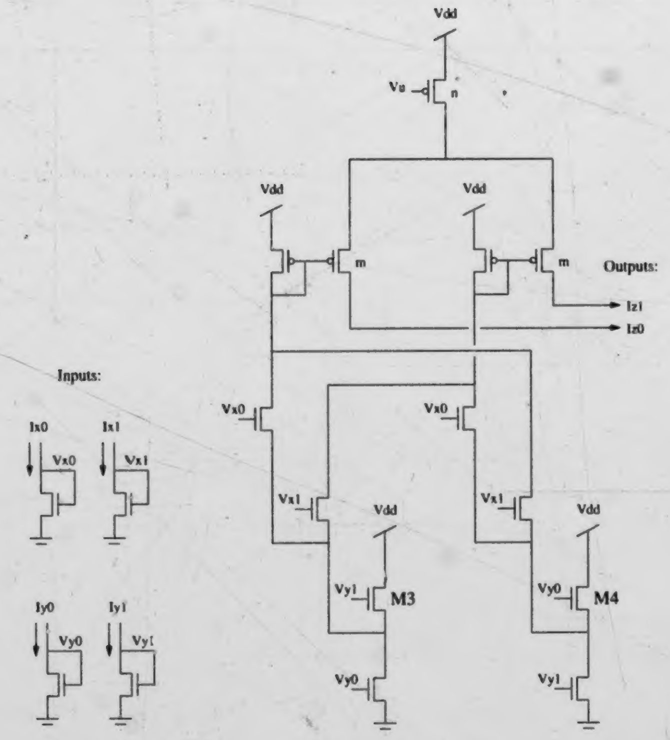

Vdd
Vu
n
Vdd
Vdd
m
m
Outputs:
Iz1
Iz0
Inputs:
Ix0
Ix1
Vx0
Vx1
Vx0
Vx0
Vx1
Vx1
Vdd
Vdd
Iy0
Iy1
Vy0
Vy1
Vy1
M3
Vy0
M4
Vy0
Vy1

Figure 6.3.3: Low-voltage check-node circuit for LDPC decoding.

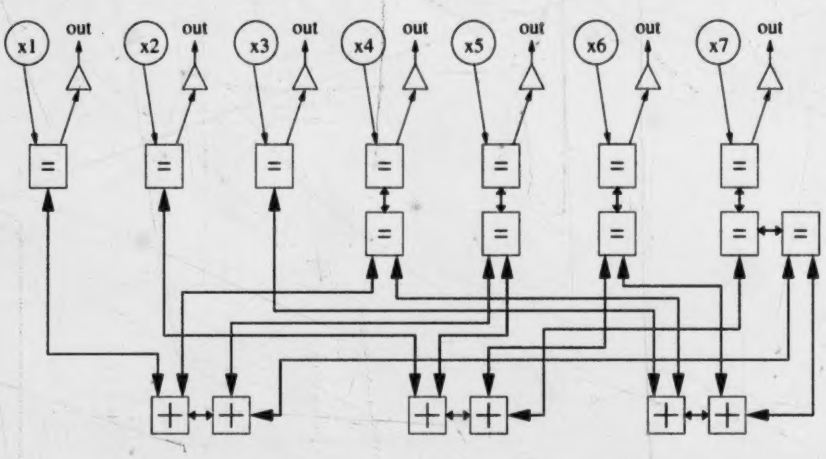


Figure 6.4.1: A Tanner graph decoder for the (7,4) Hamming code.

obtained through the Canadian Microelectronics Corporation, with Tanner TSPICE. In Figure 6.4.3, new channel samples arrive at time 2ms. The decoder begins decoding immediately. Individual bit-probabilities flip to their final decisions at about 1μs. After 3μs, the decoder's outputs have converged to their final probability estimates. The decoder achieves a throughput of 1.3 Mbit/sec. The power consumption in the decoder itself, omitting SH circuits, gain stages, and comparators, is $7.8\mu W$ at a supply of 0.6V.

The decoder thus consumes 6pJ of energy per bit. For a larger decoder (i.e. one with a larger block length), the throughput grows in proportion to the decoder's size. The power consumption also grows linearly with size. The energy per bit therefore should not change dramatically for a large, powerful iterative analog decoder.

For comparison, a recently published low-power digital LDPC decoder consumes a minimum of 1.38nJ per bit [18], with a maximum of 3.8nJ per bit at a supply of 1.5V. The low-voltage topology therefore requires approximately two orders of magnitude less energy per bit over the best known digital designs.

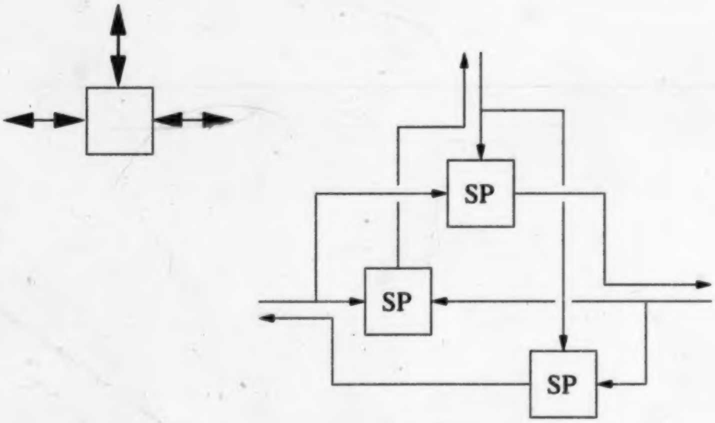


Figure 6.4.2: A complete node implementation, with a separate sum-product circuit for each edge.

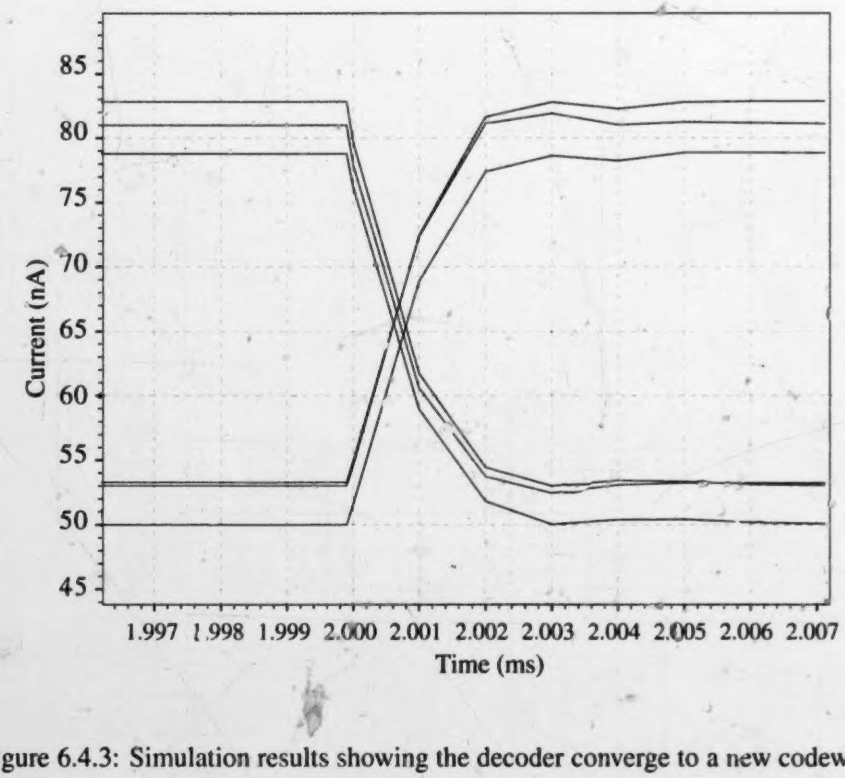


Figure 6.4.3: Simulation results showing the decoder converge to a new codeword decision.

No text

Chapter 7

*Scaling and Performance in Analog Iterative Decoders.

Analog sum-product circuits show great promise for the implementation of powerful but complex iterative error control decoders. As discussed in Chapter 2, the strength of an iterative decoder is a function of its block length. As a rule of thumb, the larger the block length, the better the code. But large codes result in large decoding circuits, with many nodes and complex routing.

Analog decoders have been demonstrated with steadily increasing size. The largest analog decoders to date include a Turbo decoder with a coded length of 120 bits [45], and a Block Product decoder of length 256, presented in Chapter 10 of this thesis. The successful implementation of these decoders is encouraging. There are, however, lingering questions about the scalability of analog decoders.

The principle questions are those of interfacing and parasitic or non-ideal analog phenomena. From the system perspective, a decoder must coexist with many other components in a complete receiver design. A decoder must have a set of interfaces which is suitable for inclusion in a real receiver system. Typically, these include analog sample storage for serial-to-parallel data conversion, and an array of comparators.

It remains to be seen whether there are limits in the number of analog samples which can be reliably stored. It must also be demonstrated that these interfaces do not cause unacceptable distortion in the input samples, that their yield can be guaranteed in a large-scale design, and that their size and power consumption do not grow at a higher rate than that of the decoder. We attempt to answer some of these questions in Section 7.1.

Scalability of analog decoders may also be limited by the influence of non-ideal characteristics in the decoding circuits themselves. The chief non-ideal characteristic of analog circuits is *mismatch*. Mismatch refers to tiny, random parametric differences between transistors. Many simulations have indicated that mismatch should have little effect on the performance of various particular (usually small) analog decoders.

One might speculate that, in a large decoder for a strong code, mismatch becomes equivalent to some additional channel noise. One might just as well speculate that the constant internal errors due to mismatch will only grow worse in a very large decoder. This accumulation of error could make it impossible for a large decoder to function at all. We address this question in Section 7.2, where we present performance results, based on density evolution, for arbitrarily large decoders over a range of mismatch conditions.

7.1 Interface architecture.

In a sophisticated communication system, several components may precede the decoder in a receiver design. If the receiver consists only of a demodulator which outputs analog log-likelihood ratios, then an analog decoder can be directly used. If other processing must occur prior to decoding, then there are two options.

First, we might perform all preprocessing stages using analog circuits. This is an attractive option in principle, in that it eliminates the need for an analog-to-digital converter in the receiver. This suggestion may not be welcome if a receiver system design is already in place, and all that is wanted is a powerful decoder.

A second approach allows a more graceful integration between the analog and digital domains. We may simply insert a digital-to-analog converter at the front of a sample-and-hold (S/H) array. If a powerful code is to be used, then the gains achieved through analog decoding outweigh the additional burden of a DAC.

Some analog decoders have employed DACs at the input, using a separate

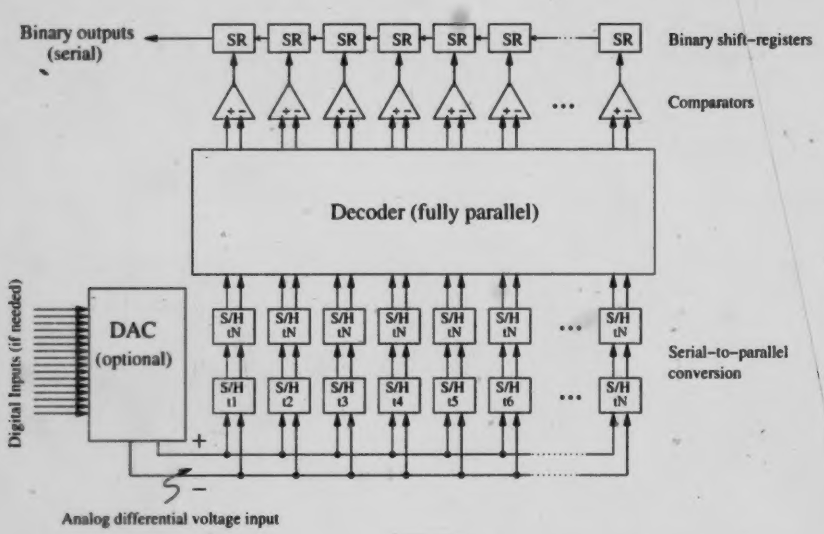

Binary outputs (serial)
SR
Binary shift-registers
Comparators
Decoder (fully parallel)
Digital Inputs (if needed)
DAC (optional)
S/H tN
S/H t1
S/H t2
S/H t3
S/H t4
S/H t5
S/H t6
Serial-to-parallel conversion
+
−
Analog differential voltage input

Figure 7.1.1: Analog decoder interface architecture.

minimally-designed DAC for each channel sample. This approach exhibited poor results [53, 44]. A single well-designed, high-quality DAC is a far superior solution, allowing analog decoders to be directly inserted in an existing receiver design [45].

A complete architecture for analog decoding is shown in Figure 7.1.1. The inputs are first converted into analog differential voltages, which correspond to LLRs. These analog voltages are then loaded step-by-step into an array of S/H registers. When a complete block is received, all samples are loaded into a second stage of S/H registers, which store the analog information for decoding. After decoding is finished, the analog outputs are presented to an array of comparators, whose binary decisions are forwarded to a bank of shift-registers. The decoded results are output serially from the head of the shift-register chain.

7.1.1 Sample-and-hold input interfaces.

Information is most often communicated serially across a communication channel. Analog decoders most often decode all bits in parallel. Channel samples must therefore be stored as they arrive, so that they may all be presented together for decoding. The clear solution, used by several designs, is to employ an array of Sample-and-

Hold (S/H) circuits, consisting of switched capacitors, to store incoming voltages [35, 95, 90].

A S/H input array is effective if the incoming channel information is expressed as a sequence of *differential voltages*, which correspond directly to log-likelihood ratios, as explained in Section 5.1.4. In this case, parasitic effects such as clock feed-through (or charge-injection) have no discernible impact on performance.

Another parasitic phenomenon – substrate leakage – is also shown to have little effect on the stored differential sample [85]. Substrate leakage does have an effect if the data is stored for a very long time (i.e. if the block length is extremely large).

For the serial-mode sample storage circuit used for the decoder in Chapter 8, implemented in a 0.5μm CMOS process, operating at a rate of 1 million samples per second, it is possible for millions of samples to be stored before the decoder's performance is adversely affected [85]. It is therefore possible to use a S/H input interface for an analog decoder with a fairly large block length.

The achievable block length for a serial analog storage array is a function of physical process parameters and operating speed. In this section, we examine the limits imposed on block length and throughput due to various process parameters.

The S/H circuit.

A simple differential S/H circuit, shown in Figure 7.1.2, suffices to store incoming analog samples. This circuit uses CMOS transmission gates as switches. The two S/H stages may be isolated by a unity-gain buffer, shown in Figure 7.1.2. The structure of a typical *transmission gate*, using pass transistors which act as simple switches, is detailed in Figure 7.1.4.

The stages may also be directly connected in an unbuffered configuration, as shown in Figure 7.1.3. In the unbuffered case, the stored voltage in the second stage is equal to half that of the first stage. Thus the magnitude and common mode of V_{in} are multiplied by 0.5 before they reach the differential pair.

The interface accepts differential voltage inputs which represent log-likelihood ratios (LLRs). The LLR format is commonly provided by analog receiver front-end circuits. LLRs, represented by differential voltages, are converted into probability

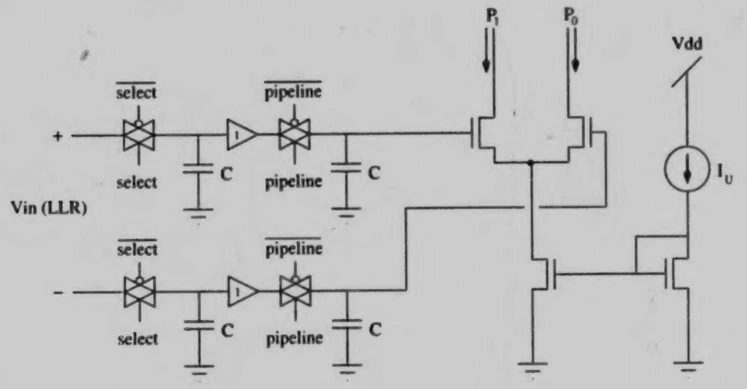

P1
P0
Vdd
select
pipeline
+
select
C
pipeline
C
IU
Vin (LLR)
select
pipeline
-
select
C
pipeline
C

Figure 7.1.2: Buffered cascade of S/H circuits, with differential pair.

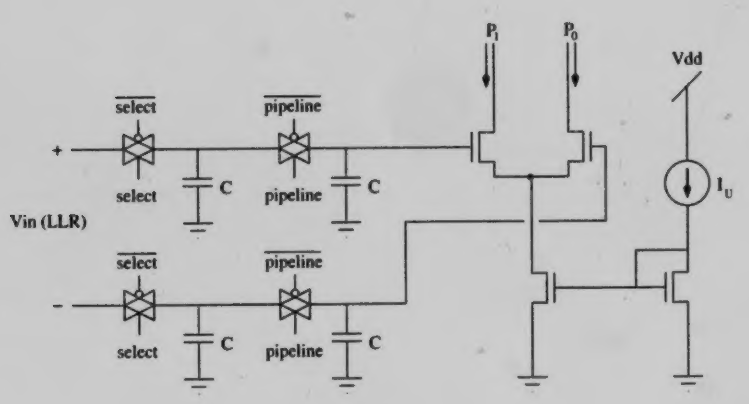

P1
P0
Vdd
select
pipeline
+
select
C
pipeline
C
IU
Vin (LLR)
select
pipeline
-
select
C
pipeline
C

Figure 7.1.3: Unbuffered S/H interface circuit.

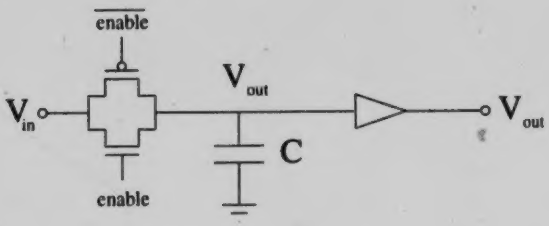

enable
V out
V in
V out
C
enable

Figure 7.1.4: S/H circuit showing the transmission gate.

currents by a differential pair biased in weak inversion. If the differential input $V_X = s \cdot X$ for a suitable scaling constant s with units $\frac{\mathrm{V}}{\mathrm{LLR}}$, and all transistors are in saturation, then the current outputs are approximately

$$\begin{aligned} P_1 &= I_U \frac{e^X}{1+e^X} && (7.1.1) \\ P_0 &= I_U \frac{e^{-X}}{1+e^{-X}}. && (7.1.2) \end{aligned}$$

When the differential pair is biased in strong-inversion, the fit to equations (7.1.1) and (7.1.2) is less exact. An approximate fit is obtained by adjusting s.

In weak-inversion, the scaling factor is $s = \frac{U_t}{\kappa} \approx .04 \frac{\mathrm{V}}{\mathrm{LLR}}$. In moderate- and strong-inversion the best fit must be found by simulation for each I_U. The best-fit scaling factor in strong-inversion is approximately linearly proportional to I_U.

Duration of storage.

The scalability of an S/H serial-to-parallel conversion circuit is limited by how long a sample can be stored [85]. The physical phenomenon of substrate leakage causes a small, nearly constant current to flow through the drain of the switch transistors when they are turned off. This causes a steady drop $\Delta V_C(t)$ in the voltage V_C stored on capacitor C:

$$\Delta V_C(t) = t \cdot \frac{I_{\text{leak}}}{C}. \qquad (7.1.3)$$

The current I_{leak} is expected to be roughly constant, independent of V_C. This is because substrate leakage is caused by an effective reverse-biased diode between the drain and substrate of a transistor, as indicated in Figure 7.1.5. The physical origin of substrate leakage is indicated for an NMOS transistor in Figure 7.1.6. The drain of the device is doped with N-type material, while the substrate is P-type. This makes a reverse-biased PN junction.

Because analog samples are stored differentially, some loss in V_C is acceptable. The amount of permissible loss, $V_{\max}$, is established by the dynamic range of the differential pair circuit on the right in Figure 7.1.2, and by the common-mode volt-

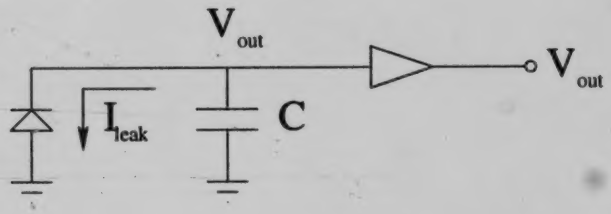

V_{out}
I_{leak}
C
V_{out}

Figure 7.1.5: Model of leakage current in S/H circuits.

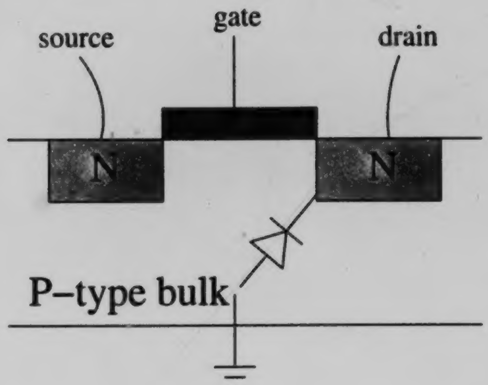

gate
source
drain
N
N
P−type bulk

Figure 7.1.6: Physical origin of substrate leakage.

age of input samples. This gives a maximum storage time of

$$t_{\max} = \frac{C \cdot V_{\max}}{I_{\text{leak}}}. \tag{7.1.4}$$

Samples arrive serially at the S/H circuit, with a frequency f_s. The minimum allowable f_s is $1/t_{\max}$. The maximum possible f_s depends on the capacitance C, and on the series-resistance R_{on} of the pass-transistor. The time needed to store each sample, τ_s, must satisfy

$$\tau_s \geq 20 \cdot R_{\text{on}} C, \tag{7.1.5}$$

where the factor of 20 has been added to ensure a conservative estimate.

The maximum number of samples that can be stored, N_s, therefore satisfies

$$N_s \leq \frac{t_{\max}}{\tau_s} = \frac{V_{\max}}{20 \cdot R_{\text{on}} I_{\text{leak}}}. \tag{7.1.6}$$

For an AMI 0.5μm CMOS process, I_{leak} has been measured at $\sim 10^{-17} A$ [96]. If $R_{\text{on}} \sim 10k\Omega$ and $V_{\max} = 0.01V$, then $N_s \sim 10^9$ bits.

Note that (7.1.6) has no dependence on C. The choice of C depends only on the desired sample frequency. In general, C should be made as large as possible to meet the desired speed.

Equation 7.1.6 can be further simplified if we account for the dependence of R_{on} and I_{leak} on device dimensions. The channel resistance is inversely proportional to the width, W, of the pass transistor. The leakage current is directly proportional to W. Assuming a minimum-sized NMOS pass transistor, we arrive at

$$\begin{aligned} R_{\text{on}} &= \frac{L_{\min}}{\mu_n C_{\text{ox}} W \left(V_{\text{gs}} - V_{\text{th}}\right)} \\ I_{\text{leak}} &= k_2 W \\ \Rightarrow N_s &\leq V_{\max} \frac{\mu_n C_{\text{ox}} \left(V_{\text{dd}} - V_{\text{CM}} - V_{\text{th}}\right)}{20 \cdot L_{\min} k_2}. \end{aligned} \tag{7.1.7}$$

This result expresses the sample storage limit mainly in terms of fixed process parameters. The voltage V_{CM} represents the common-mode offset of the stored channel samples. The average V_{gs} for pass transistors is equal to $V_{\text{dd}} - V_{\text{CM}}$, which is substituted to obtain the final form of (7.1.7).

Most of the process constants in (7.1.7) are precisely known for a given process. The leakage constant, k_2, is often not well modeled or measured, and may need to be directly measured to verify the suitability of a S/H interface in a chosen process.

Charge-injection.

Charge-injection is a significant source of distortion in S/H circuits [79]. When a CMOS transmission gate is switched off, the channel charge of the transistor(s) is expelled and a portion of it, ΔQ, is deposited on the capacitor C.

ΔQ is signal-dependent, so that, if the voltage stored on the capacitor is V_C, then $\Delta Q = f(V_C)$. While f is in general a non-linear function, it is sometimes appropriate to approximate it by a linear function $\Delta Q \approx k \cdot V_C$. This approximation is especially appropriate when the variation in scaled LLR inputs is small (less than 100mV).

Let $V_{in} = V_{in}^{+} - V_{in}^{-}$ refer to the scaled-LLR differential input to the S/H circuit, and let $V_{out} = V_{out}^{+} - V_{out}^{-}$ refer to the output. The circuit's output after charge-injection is approximately

$$V_{out} = V_{in}\left(1 + \frac{k}{C}\right). \tag{7.1.8}$$

As shown by (7.1.8), the effect of charge-injection is approximately equivalent to multiplication of V_{in} by a constant slightly greater than one. Bit error rate simulations of several decoders indicate that performance is not affected when inputs are scaled by factors as large as 2 and as small as 0.5. This is not a surprising result; the textbook MAP decoding procedure for BPSK signals on the AWGN channel, based on minimum-Euclidean distance, is completely invariant under scaling by a constant factor.

7.1.2 Comparators and yield.

A complete analog decoding architecture must employ one or more comparators to convert the analog soft outputs into digital decisions. It is somewhat easier to design a good low-speed, low-energy decoder than a high-speed one, so a large array of output comparators is preferred. All comparators exhibit an unwanted input *offset voltage*, which is mainly due to mismatch.

There are several techniques for compensating offset in comparator designs [8, 93], but for analog decoders we are interested in expending minimum resources on the comparator. In a typical analog decoder with block length n, there are n comparators in the output stage. If the comparator design is not very small, then the expense of the output stage may overtake the benefits of implementing an analog decoder. On the other hand, a very simple, low-energy comparator design may exhibit a large variance of offset voltages. We now address the effect this will have on performance.

Performance loss due to comparator faults.

As in the Density Evolution analysis of Section 4.2, we employ the Gaussian approximation, in which we assume that the log-likelihood ratio at the decoder's output, X, is Gaussian distributed, and that its mean is proportional to its variance. We also assume that the offset voltage is Gaussian distributed with zero mean.

Because a differential voltage is directly proportional to a log-likelihood ratio, we may speak in terms of the comparator's offset log-likelihood, X_o. We further assume that no comparator's LLR offset exceeds a specified limit, L, which is chosen to be a fraction of the decoder's maximum possible output.

Based on these assumptions, we may say that the zero-offset error probability, P_e, is given by

$$P_e = Q\left(-\frac{\sigma_X}{2}\right), \tag{7.1.9}$$

where $Q()$ is the well-known Gaussian error integral function. When the offset is accounted for, we find that the error probability with offset, $P_e(X_o)$, is given by

$$P_e(X_o) = \int_{-L}^{L} \mathcal{G}\left(\frac{X_o}{\sigma_o}\right) \cdot Q\left(\frac{X_o}{\sigma_X} - \frac{\sigma_X}{2}\right) dX_o, \tag{7.1.10}$$

where $\mathcal{G}()$ is the Normal density function.

Figure 7.1.7 displays the ratio $\frac{P_e(X_o)}{P_e}$ as a function of offset standard deviation (expressed in the LLR domain). Results are presented for several zero-offset BERs. If the offset deviation is less than two, then the BER is increased by a constant factor independent of SNR. It therefore seems that a large range of offsets are acceptable.

In practice, the offset deviation depends on a variety of factors including the exact circuit design, the fabrication technology, and the signal scaling between LLRs and differential voltages. These factors can be evaluated for any particular design.

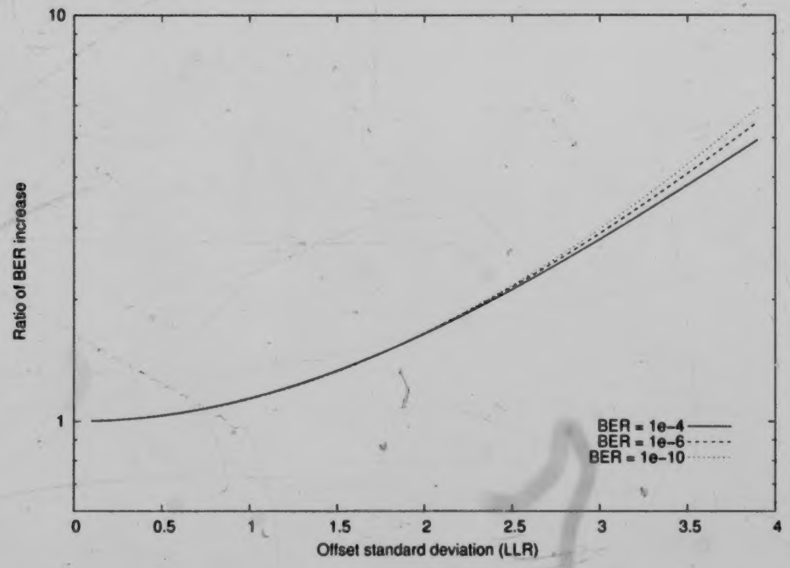


Figure 7.1.7: The increase in error probability due to comparator offsets. Offsets are assumed to be Gaussian distributed voltages, represented in the Figure in normalized log-likelihood units.

Note that a small voltage offset can correspond to a large LLR offset if the scaling parameter (s in Section 7.1.1) is small.

Device yield.

In general, as Figure 7.1.7 demonstrates, the limiting concern in comparator design is not performance but *yield*. Suppose we want a failure rate of one chip per thousand, and each chip has one thousand comparators. We must achieve a comparator failure rate better than one in a million. Because we stipulate that no comparator's offset magnitude can exceed L, we require some means of verifying that this condition is satisfied in a fabricated design.

It is conceivable that an analog decoder design can employ a *built-in self-test* (BIST) in which any offset magnitude greater than a specified limit L is detected, and such chips are discarded as failures. A suitable BIST may or may not be feasible for an analog decoding architecture. This thesis does not present a BIST solution for comparators in analog decoders. Because of the sensitivity between offset and yield, we suggest that a comparator BIST solution is necessary for verification of analog decoders when manufactured in bulk.

The decoder determines L, and the comparator design must be adjusted accordingly to meet yield requirements. An analog decoder design is only as good as the comparators which resolve its output into useful digital information.

7.2 Mismatch.

By far the biggest concern about the performance of large-scale analog decoders is device mismatch. In this section we examine mismatch and demonstrate that, on a large scale, mismatch can be dealt with as a kind of noise. We conclude that two types of mismatch effects exist: feed-forward and lateral.

Definition 7.2.1. Feed-Forward processing. Circuits, especially the input stages, in an analog decoder in which signals flow from input to output, with little or no connection to neighboring circuits. In feed-forward circuits, mismatch can be straightforwardly referred to the channel's output and treated as an equivalent increase in channel noise [24].

Definition 7.2.2. Lateral processing. Circuits for which the outputs are dependent on input from several adjacent cells; the iterative parts of a decoder. In these highly-connected processing stages, mismatch cannot be straightforwardly referred to the channel.

It will be shown in Section 7.2.3 that a separate lateral effect exists. This effect is severe when mismatch is large. When the mismatch variance is maintained at a reasonable, controlled level, however, the lateral effect is negligible in comparison to the feed-forward effect. Acceptable mismatch levels for lateral stages are attained by minimum-sized devices in many current processes [27]. We conclude that it is most important to optimize for mismatch in the *input stages* to an analog decoder.

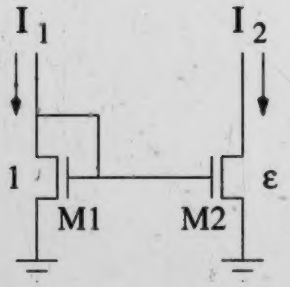


Figure 7.2.1: Current mirror circuit.

7.2.1 Modeling mismatch in analog sum-product circuits.

Mismatch is defined as a difference in the behavior of two devices which are designed to be identical. Mismatch arises from a variety of random and deterministic events during device fabrication [65, 27]. Deterministic mismatch components can typically be neglected for devices which are small and closely spaced. The remaining random phenomena are small, independent variations in parameters and device geometry. Because mismatch variations are small, they can be considered additive. The central limit theorem therefore applies, and the complete effect of mismatch is expressed as a Gaussian-distributed random variation in one or more device parameters.

An important building block for analog circuits is the current mirror, shown in Figure 7.2.1. Ideally for this circuit, we should find that $I_2 = I_1$. In reality, mismatch between devices M1 and M2 results in a random variation ε, so that

$$I_2 = I_1 \cdot (1+\varepsilon) \tag{7.2.1}$$

where ε is a zero-mean Gaussian random variable. In this circuit, I_1 is considered to be the input, and I_2 the output.

This model is quite common in the semiconductor literature [27, 55, 65], though ε cannot be considered a true Gaussian. The tail of its distribution, where $(1+\varepsilon) < 0$, is not physically possible (it violates conservation of energy). If the probability density function for $(1+\varepsilon)$ is $\rho_\varepsilon(1+\varepsilon)$, then we must require

$$\rho_\varepsilon(1+\varepsilon) = 0, \forall \varepsilon < -1$$

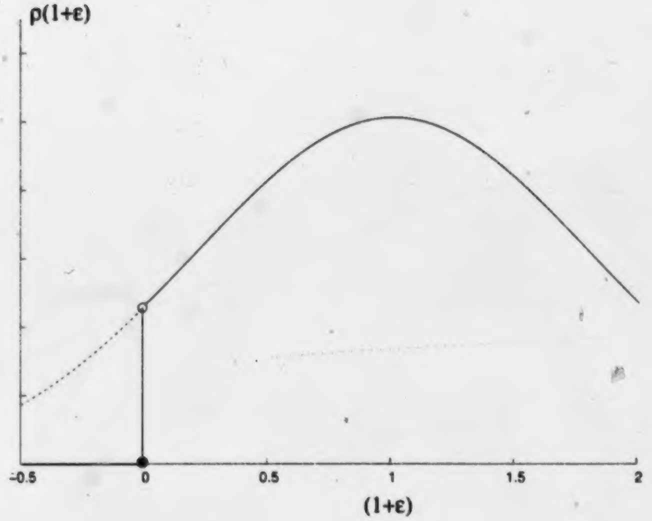


Figure 7.2.2: Density function of the mismatch factor (with exaggerated variance).

(the resulting density function must, of course, be appropriately normalized). What happens to the physical density function as $(1+\varepsilon)$ approaches zero is unclear. Very little experimental research is available on the tails of the mismatch distribution.

When $(1+\varepsilon)=0$, the device is completely destroyed and no current will flow. This is a common failure event in large-scale digital circuits. Large chips are typically designed with a test procedure which can detect the presence of any failed devices. After fabrication, any chips which fail this test are discarded. If we assumeroon that such a test is available for analog decoder chips, then we may safely exclude the complete failure case where $(1+\varepsilon)=0$ from our analysis. The resulting physically plausible density function is illustrated in Figure 7.2.2.

We restrict our analysis to the canonical Gilbert multiplier sum-product circuit, shown in Figure 7.2.3. This circuit is repeated to construct an iterative decoder for any LDPC code [52, 31]. If the devices were perfectly matched, this circuit would produce outputs given by (5.1.11).

To account for the effect of mismatch, we must modify the translinear principle (Section 5.1) by applying the $(1+\varepsilon)$ factor to the affected currents. Applying this modified translinear principle on the two distinct loop topologies in Figure 7.2.3,

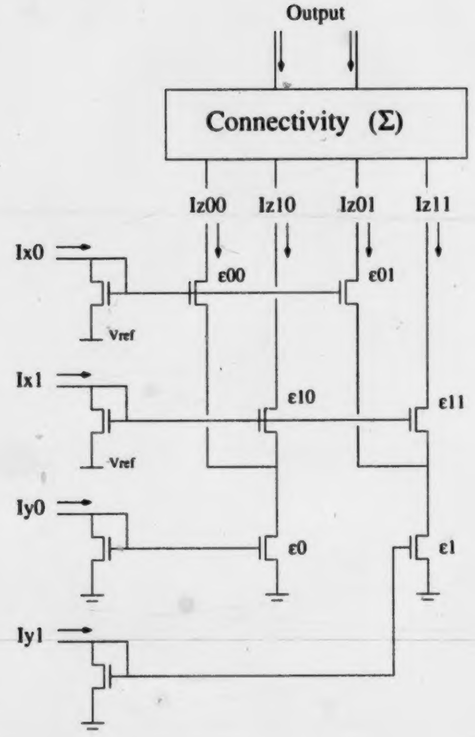

Output
Connectivity (Σ)
Iz00
Iz10
Iz01
Iz11
Ix0
Ix1
Iy0
Iy1
Vref
Vref
ε00
ε01
ε10
ε11
ε0
ε1

Figure 7.2.3: Basic Gilbert multiplier sum-product circuit.

we arrive at the following equations:

$$\begin{aligned} Ix_0 \cdot Iz_{1j}(1+\varepsilon_{1j}) &= Ix_1 \cdot Iz_{0j}\left(1+\varepsilon_{0j}\right) \\ \Rightarrow Iz_{\bar{i}j} &= \tfrac{Ix_{\bar{i}}}{Ix_i} \cdot \tfrac{\left(1+\varepsilon_{ij}\right)}{\left(1+\varepsilon_{\bar{i}j}\right)} \cdot Iz_{ij} \end{aligned} \tag{7.2.2}$$

$$\begin{aligned} Iz_{0j} + Iz_{1j} &= Iy_j\left(1+\varepsilon_j\right) \\ \Rightarrow Iy_j\left(1+\varepsilon_j\right) &= Iz_{ij} + \tfrac{Ix_{\bar{i}}}{Ix_i} \cdot \tfrac{\left(1+\varepsilon_{ij}\right)}{\left(1+\varepsilon_{\bar{i}j}\right)} \cdot Iz_{ij}. \end{aligned} \tag{7.2.3}$$

$$\Rightarrow \ Iz_{ij}\,(\text{actual}) = \frac{Ix_i \cdot Iy_j \cdot \left(1+\varepsilon_j\right)\left(1+\varepsilon_{\bar{i}j}\right)}{Ix_i\left(1+\varepsilon_{\bar{i}j}\right) + Ix_{\bar{i}}\left(1+\varepsilon_{ij}\right)}. \tag{7.2.4}$$

The result (7.2.4) is the circuit's output corrected for mismatch. Our model of mismatch is almost identical to that used in [55].

7.2.2 Feed-forward analysis.

The feed-forward analysis of mismatch was introduced by Jie Dai [24], and is based on a sequence of Taylor approximations to simplify (7.2.4). We also neglect correlation between the numerator and denominator of (7.2.4). The important approximations are as follows:

$$\begin{aligned} a\varepsilon_1 + b\varepsilon_2 &= \varepsilon\sqrt{a^2+b^2} \\ \frac{1}{1+\varepsilon} &\approx 1+\varepsilon \\ \varepsilon_1\varepsilon_2 &\approx 0 \\ \ln(1+\varepsilon) &\approx \varepsilon. \end{aligned}$$

We illustrate mismatch referral for a differential-pair input circuit, as shown in Figure 7.2.4. In a decoder with a large block length, there are many instances of this differential pair circuit which are independent of each other. We may therefore speak of the mismatch as noise.

We first observe that, for $I_0 = I_1$, we should have an input (ideally) of zero. In this case, we note that

$$\begin{aligned} V_Y &= \ln\left(\frac{1+\varepsilon_1}{1+\varepsilon_2}\right) \\ &\approx \varepsilon\sqrt{2}, \end{aligned} \tag{7.2.5}$$

where we have used the normalized units of voltage and current, as used in (5.2.5) in Section 5.2.1.

When using normalized units, the scaling factor s (explained in Section 7.1.1) is equal to one. To refer V_Y to the channel's output, we write the standard deviation of the channel's noise as $\sigma_n^2 = N_0/2$, and the effective variance due to mismatch as σ_m^2, so that the total effective noise, inclusive of mismatch, is

$$\begin{aligned} \sigma_{\text{tot}}^2 &= \sigma_n^2 + \sigma_m^2 \\ &= \frac{N_0}{2} + \frac{2N_0^2\sigma_\varepsilon^2}{16} \\ &= \frac{N_0}{2}\left(1 + \frac{N_0\sigma_\varepsilon^2}{4}\right). \end{aligned} \tag{7.2.6}$$

By placing (7.2.6) in the log domain, we arrive at the effective loss in SNR due to mismatch:

$$\text{loss (dB)} = 10 \cdot \log_{10}\left(1 + \frac{N_0\sigma_\varepsilon^2}{4}\right). \tag{7.2.7}$$

Jie Dai found that this procedure could be applied in general to canonical CMOS sum-product circuits, arriving at the general formula

$$\text{loss (dB)} = 10 \cdot \log_{10}\left(1 + \nu \cdot N_0 \cdot \sigma_\varepsilon^2\right), \tag{7.2.8}$$

where ν is a constant, ranging from 0.25 to 8. The constant ν depends on the circuit topology, and increases as more feed-forward stages are added.

The loss due to the feed forward effect is plotted in Figure 7.2.5 for rate-1/2 codes, for which the ideal SNR = 1.5 dB. The loss due to mismatch can become quite large for moderate ν. Because ν is a fixed parameter of the circuit, it cannot usually be improved. The only way to reduce the loss due to mismatch is to reduce σ_ε, which can be accomplished by using several mismatch-optimizing layout techniques in feed-forward stages.

7.2.3 Density evolution analysis of lateral effects.

The method of density evolution provides a means of evaluating upper limits to the performance of iterative decoders for arbitrarily large LDPC codes [21, 70].

To apply density evolution to the Gilbert-multiplier decoding architecture, we must be able to compute a node's mean output message based on its mean input

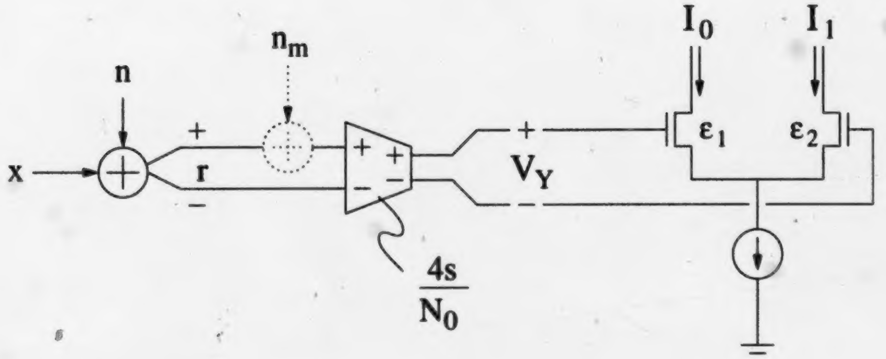


Figure 7.2.4: Mismatch referral for a differential-pair circuit. V_Y represents the offset voltage needed to make $I_0 = I_1$ when the received channel sample is $r = 0$ for a particular pair of mismatch values, ε_1 and ε_2. The effective additional channel noise, n_m, is the offset in the received sample which is needed to produce V_Y.

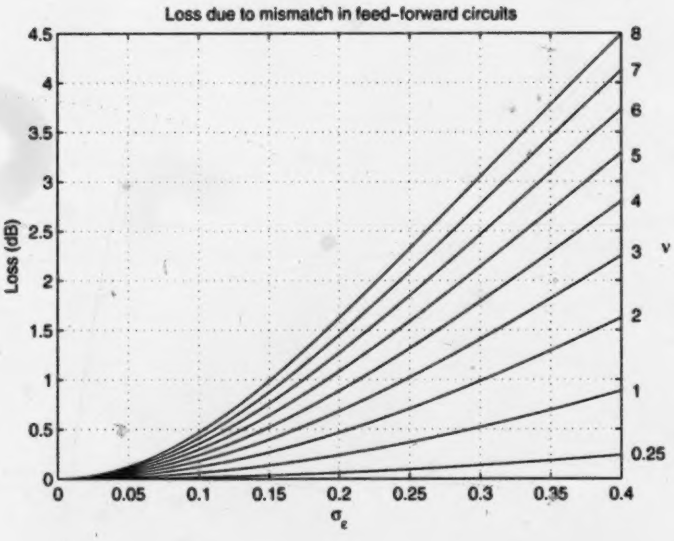


Figure 7.2.5: Performance loss due to mismatch in a feed-forward circuit, for a rate-1/2 code at $E_b/N_0 = 1.5\,\text{dB}$. The parameter ν is varied from 0.25 to 8, as indicated to the right of each curve.

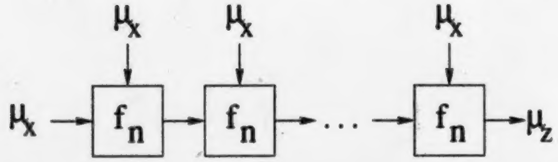


Figure 7.2.6: Illustration of iterated VEGAS integration for nodes of degree > 3.

message. Let X and Y be the input LLR messages to the analog node $\mathcal{N}$. Let $\overrightarrow{\varepsilon}$ be the vector of Gaussian-distributed mismatch parameters, each with variance σ_{ε}^2. Let μ_X and μ_Y refer to the mean of X and Y, respectively, and let σ_X^2 and σ_Y^2 be the variances of X and Y.

Let Z be the node's output message, and let $Z = f_n\left(X, Y, \overrightarrow{\varepsilon}\right)$ be the function describing the behavior of node $\mathcal{N}$. Recall that f_n is determined by (7.2.4). Then the mean of Z, μ_Z, is equal to

$$\begin{aligned}&\mu_Z = \\ &\int \cdots \int f_n\left(x, y, \overrightarrow{\varepsilon}\right) \rho\left(\frac{x-\mu_X}{\sigma_X}\right) \rho\left(\frac{y-\mu_Y}{\sigma_Y}\right) \rho_{\varepsilon}\left(\frac{\overrightarrow{\varepsilon}}{\sigma_{\varepsilon}}\right) dx\, dy\, d\overrightarrow{\varepsilon}\end{aligned} \tag{7.2.9}$$

where ρ is the zero-mean, unit-variance normal density function.

To compute this integral, we use a method of adaptive Monte Carlo integration known as the VEGAS algorithm [50] with the procedure described in Section 4.3.3 of this thesis. An implementation of the VEGAS algorithm is available in the GNU Scientific Library [1].

To determine μ_Z for a node with degree greater than three, we iterate the VEGAS integration, as illustrated in Figure 7.2.6. Each box labeled 'f_n' in Figure 7.2.6 represents an integration. Care has been taken in this iteration to ensure that the final estimate μ_Z has acceptable precision (see Section 4.3.3 of this thesis).

Using this algorithm, the threshold of a regular LDPC ensemble can be evaluated as a function $s^*(\sigma_{\varepsilon})$ of the mismatch standard deviation σ_{ε}. From [21] we know the exact thresholds s^*_{exact} for a variety of regular distributions. We therefore define a function $f_{\text{loss}}(\sigma_{\varepsilon}) = \frac{s^*(\sigma_{\varepsilon})}{s^*_{\text{exact}}}$. This function has been computed for the regular ensembles listed in Table 4.3. The results (in dB) are shown in Figure 7.2.7.

Figure 7.2.7 reveals similar exponential losses for each ensemble. We offer the conjecture that a particular decoder would experience a performance loss due

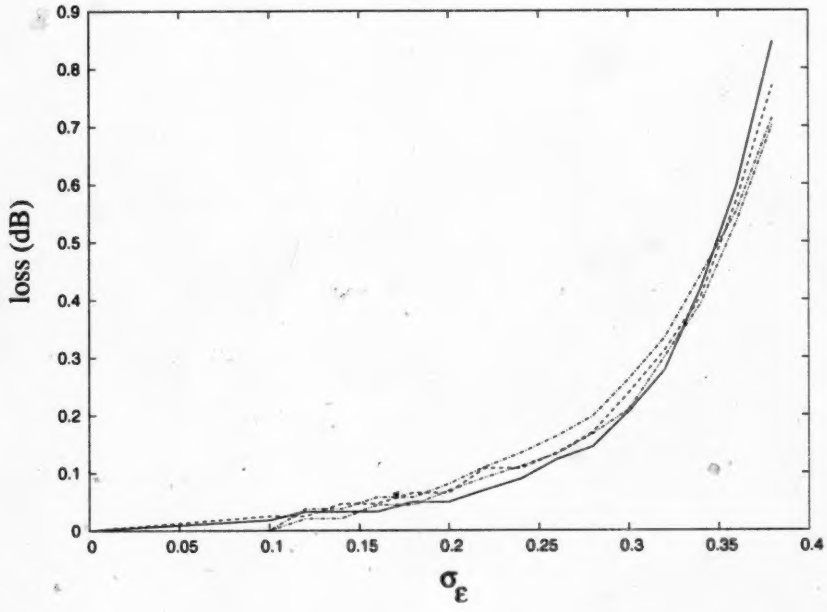


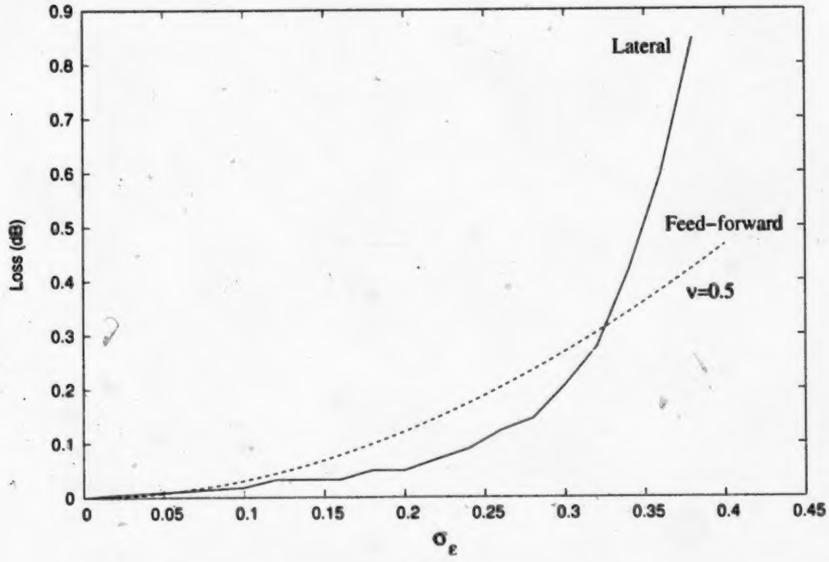
Figure 7.2.7: The threshold loss due to mismatch for regular LDPC ensembles corrseponding to those in Table 4.3.

to mismatch similar to that of the threshold itself in Figure 7.2.7. The designer may therefore use these results as a "rule of thumb" for sizing transistors. If a performance loss of 0.2 dB is tolerable, then transistors should be sized such that $\sigma_\varepsilon < 30\%$.

Even in the most aggressive present CMOS processes, mismatch does not exceed $\sigma_\varepsilon \approx 25\%$ for even near-minimum sized devices [27]. Obtaining mismatch better than 10% is easily accomplished by slightly increasing the transistor size. It should also be noted that a difference of 0.1 dB is near the error margin introduced by the Gaussian approximation itself [21]. We therefore conclude that for a sufficiently large code, no statistically significant performance loss occurs until $\sigma_\varepsilon > 0.2$.

7.2.4 Comparison of feed-forward and lateral mismatch effects.

To compare the effect of lateral and feed forward effects, a pair of representative loss curves is shown in Figure 7.2.8. The lateral loss curve is for a $(3,12)$ regular



Fi[illegible]8: Comparison of feed-forward and lateral losses.

LDPC ensemble. Recall that for an LDPC ensemble, the parameters are written as (d_v,d_c), refering to node degrees as opposed to code length. The feed-forward curve in in Figure 7.2.8 assumes $\nu = 0.5$. It is clear that the feed-forward effect dominates at low levels of mismatch, even for this small value of ν.

The lateral effect is catastrophic for high levels of mismatch. For ordinary mismatch conditions, it is essentially irrelevant. As long as mismatch in the circuit does not exceed a critical variance of about 25%, there is no danger due to the lateral effect.

The feed-forward effect, however, has a more pronounced influence in low-mismatch conditions. We therefore conclude that great emphasis must be placed on mismatch optimization in feed-forward stages, while a more relaxed approach is permissible in the lateral circuits. Input interface circuits, as part of the feed-forward section, must be optimized for low mismatch, while lateral stages can be implemented with generic "soft-gate" components as prescribed by Lustenberger [53] and Jie Dai [24].

No text

Chapter 8

*A CMOS Analog Decoder for an (8,4) Tailbiting Hamming Code.

A version of this chapter is published in Ref. [90].

In this chapter we present an analog implementation of the tailbiting (8,4) Hamming trellis decoder. The structure and algorithm for this decoder are described in Section 3.2.4. All of the necessary decoding operations are implemented using the canonical CMOS sum-product approach outlined in Section 5.2.2. We also present the design of a serial-to-parallel S/H interface based on the principles of Section 7.1.1. An array of current comparators is used to resolve the circuit's analog outputs to digital bits.

The decoder was implemented in an AMI 0.5μm digital CMOS process. All transistors in sum-product circuits are sized 2μm × 4μm, which is about four times the minimum transistor size for this process. The circuit accepts serial- or parallel-mode analog differential voltages, corresponding to log-likelihood ratios, and outputs digital bits. The output can be selected to be serial- or parallel-mode, but the serial mode is not available due to a layout fault.

8.1 The analog sum-product components.

According to Section 3.2.4, the decoder is divided into seven computational parts:

- The "Tree" which combines a pair of individual bit-probabilities into two-bit symbol probabilities.

- The "clockwise" and "counter-clockwise" computations of the first trellis section, T_1.
- The "clockwise" and "counter-clockwise" computations of the second trellis section, T_2.
- The "upward" computation of T_1 (out).
- The "upward" computation of T_2 (out).

8.1.1 The Tree circuit.

The "Tree" operation is represented by the matrix product

$$\begin{aligned}
\begin{bmatrix} \gamma^{(00)} \\ \gamma^{(01)} \\ \gamma^{(10)} \\ \gamma^{(11)} \end{bmatrix} &= \begin{bmatrix} \lambda_1^{(0)} & \\ \lambda_1^{(1)} & \\ & \lambda_1^{(0)} \\ & \lambda_1^{(1)} \end{bmatrix} \cdot \begin{bmatrix} \lambda_0^{(0)} \\ \lambda_0^{(1)} \end{bmatrix} && (8.1.1) \\
\underline{\gamma} &= Tree(\lambda_1) \cdot \underline{\lambda_0}. && (8.1.2)
\end{aligned}$$

This computation is implemented by the circuit in Figure 8.1.1, which includes renormalization.

8.1.2 Trellis section one.

The first trellis section has clockwise and counterclockwise components, represented by the matrix operations

$$\begin{aligned}
T_1(c) &= \begin{bmatrix} \gamma^{(00)} & 0 \\ \gamma^{(11)} & 0 \\ 0 & \gamma^{(01)} \\ 0 & \gamma^{(10)} \end{bmatrix} && (8.1.3) \\
T_1(cc) &= [T_1(c)]^{\mathrm{T}}.
\end{aligned}$$

The circuit for $T_1(c)$ is shown in Figure 8.1.2, and that for $T_1(cc)$ in Figure 8.1.3. These circuits also show the renormalization portion.

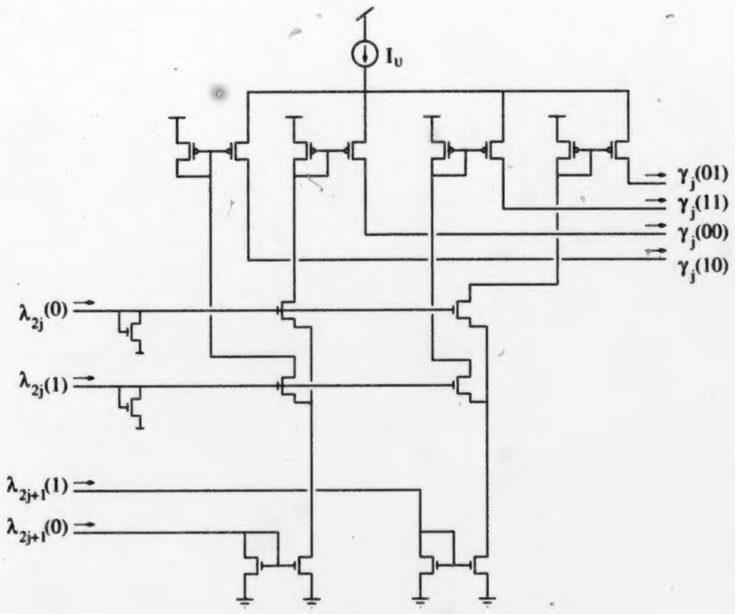

I_U
$\gamma_j(01)$
$\gamma_j(11)$
$\gamma_j(00)$
$\gamma_j(10)$
$\lambda_{2j}(0)$
$\lambda_{2j}(1)$
$\lambda_{2j+1}(1)$
$\lambda_{2j+1}(0)$

Figure 8.1.1: Circuit for the "Tree" computation.

8.1.3 Trellis section two.

The matrix operations for the second trellis section are

$$\begin{aligned} T_2(c) &= \begin{bmatrix} \gamma^{(00)} & \gamma^{(11)} & \gamma^{(10)} & \gamma^{(01)} \\ \gamma^{(11)} & \gamma^{(00)} & \gamma^{(01)} & \gamma^{(10)} \end{bmatrix} \\ T_2(cc) &= [T_2(c)]^{\mathrm{T}}. \end{aligned} \qquad (8.1.4)$$

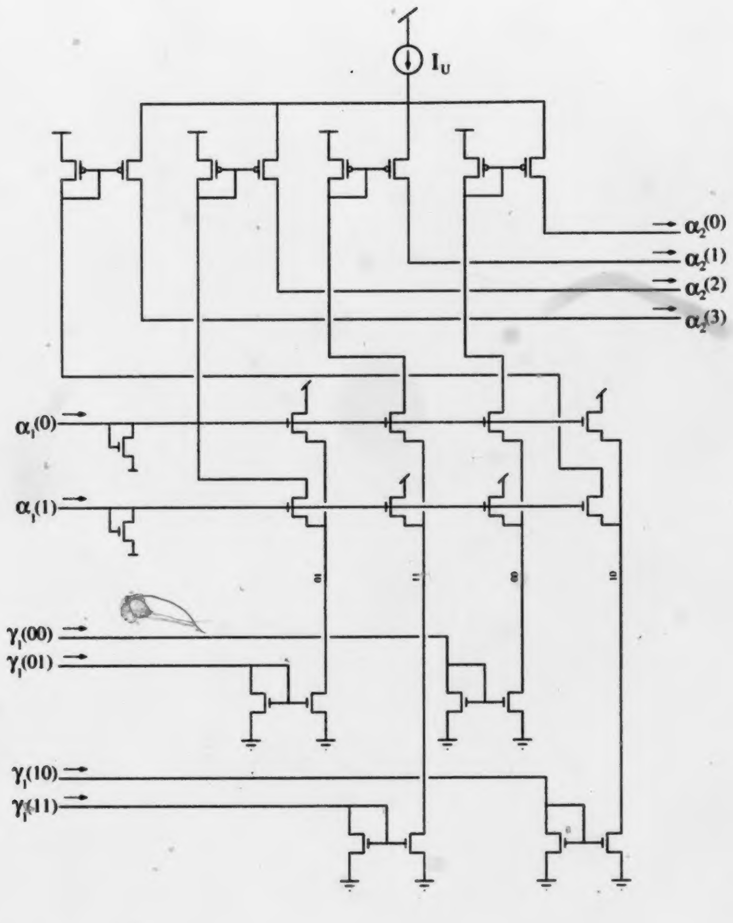
The circuit for $T_2(c)$ is shown in Figure 8.1.4, and that for $T_2(cc)$ in Figure 8.1.5. These circuits also show the renormalization portion.

8.1.4 T_1 (out).

The T_1 (out) computation is represented by the matrix operation

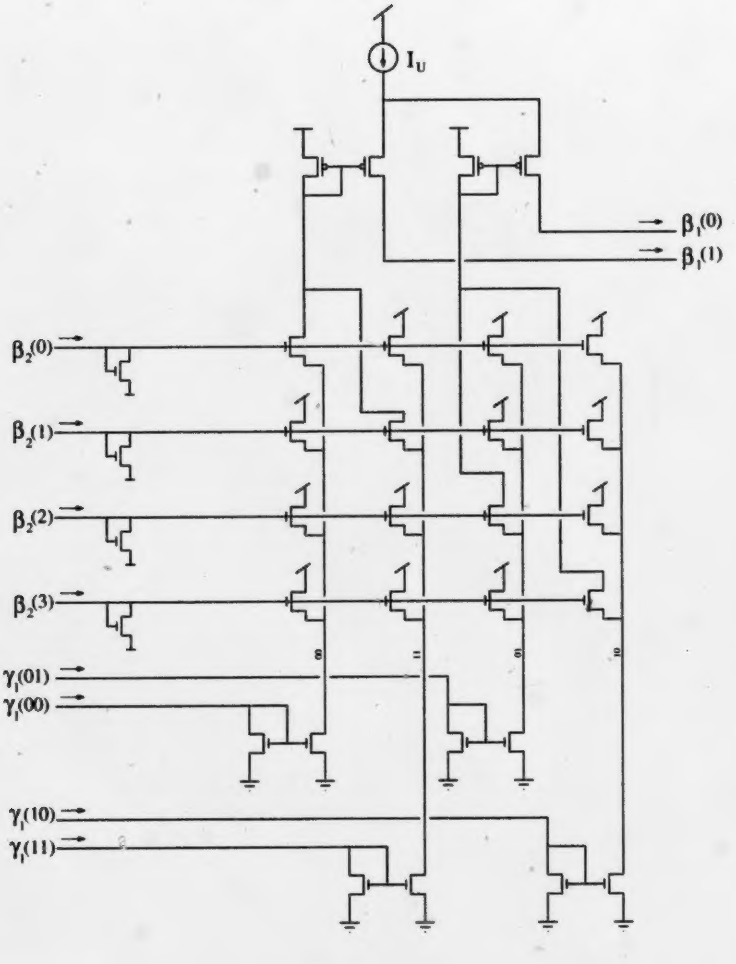
$$T_1(out) = \begin{bmatrix} \beta^{(0)} & 0 \\ 0 & \beta^{(1)} \end{bmatrix}. \qquad (8.1.5)$$

The circuit for this computation is shown in Figure 8.1.6.



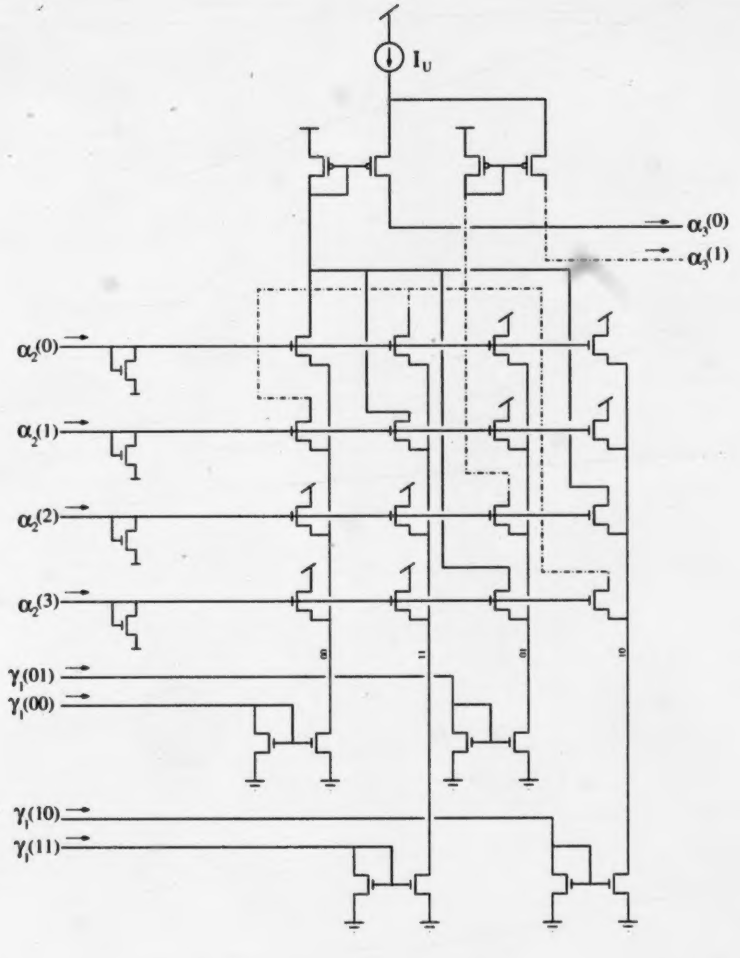
I_U
$\alpha_2(0)$
$\alpha_2(1)$
$\alpha_2(2)$
$\alpha_2(3)$
$\alpha_1(0)$
$\alpha_1(1)$
01
11
00
10
$\gamma_1(00)$
$\gamma_1(01)$
$\gamma_1(10)$
$\gamma_1(11)$

Figure 8.1.2: Circuit implementation of $T_1(c)$.



I_U
β_1(0)
β_1(1)
β_2(0)
β_2(1)
β_2(2)
β_2(3)
00
11
01
10
γ_1(01)
γ_1(00)
γ_1(10)
γ_1(11)

Figure 8.1.3: Circuit implementation of T_1 (*cc*).



I_U
$\alpha_3(0)$
$\alpha_3(1)$
$\alpha_2(0)$
$\alpha_2(1)$
$\alpha_2(2)$
$\alpha_2(3)$
00
11
01
10
$\gamma_1(01)$
$\gamma_1(00)$
$\gamma_1(10)$
$\gamma_1(11)$

Figure 8.1.4: Circuit implementation of $T_2(c)$.

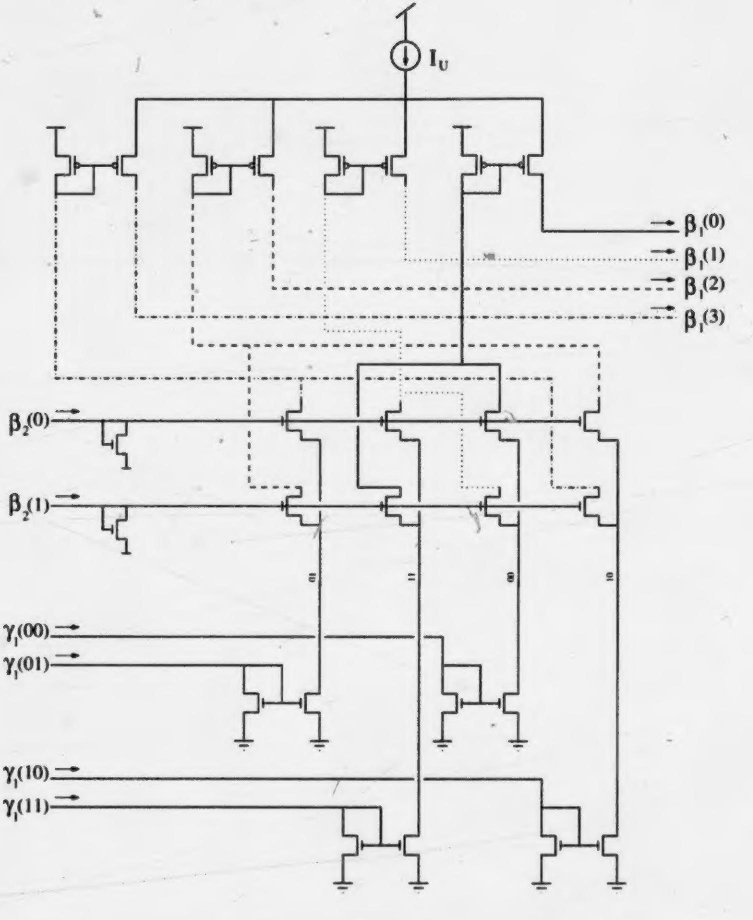
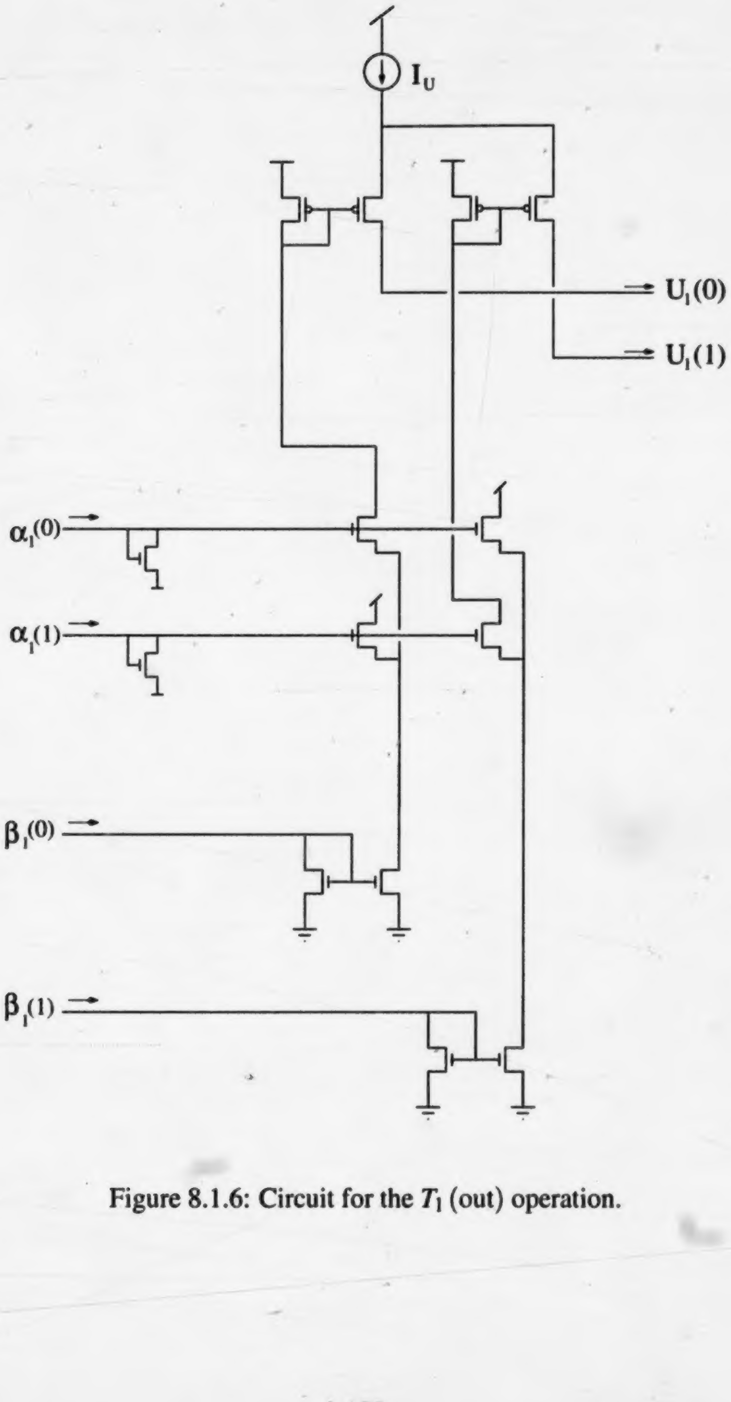

I_U
β1(0)
β1(1)
β1(2)
β1(3)
β2(0)
β2(1)
01
11
00
10
γ1(00)
γ1(01)
γ1(10)
γ1(11)

Figure 8.1.5: Circuit implementation of $T_2\,(cc)$.



I_U
$U_i(0)$
$U_i(1)$
$\alpha_i(0)$
$\alpha_i(1)$
$\beta_i(0)$
$\beta_i(1)$

Figure 8.1.6: Circuit for the T_1 (out) operation.

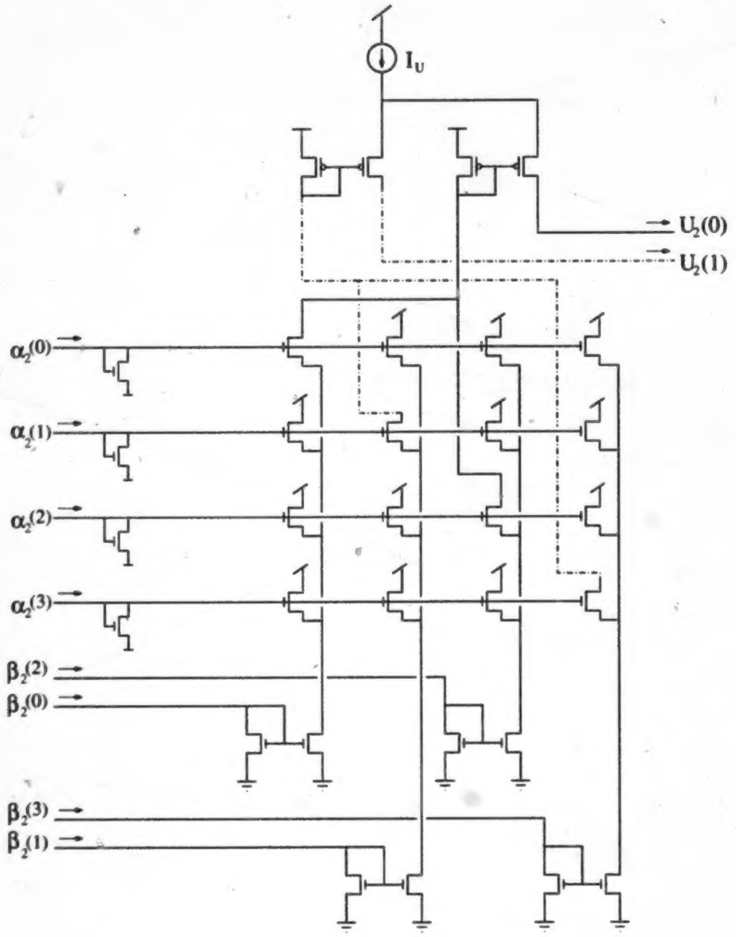

I_U
$U_2(0)$
$U_2(1)$
$\alpha_2(0)$
$\alpha_2(1)$
$\alpha_2(2)$
$\alpha_2(3)$
$\beta_2(2)$
$\beta_2(0)$
$\beta_2(3)$
$\beta_2(1)$

Figure 8.1.7: Circuit for the T_2 (out) operation.

8.1.5 T_2 (out).

The T_2 (out) computation is represented by the matrix operation

$$T_2\,(out) = \begin{bmatrix} \beta^{(0)} & 0 & \beta^{(2)} & 0 \\ 0 & \beta^{(1)} & 0 & \beta^{(3)} \end{bmatrix}. \tag{8.1.6}$$

The circuit for this computation is shown in Figure 8.1.7.

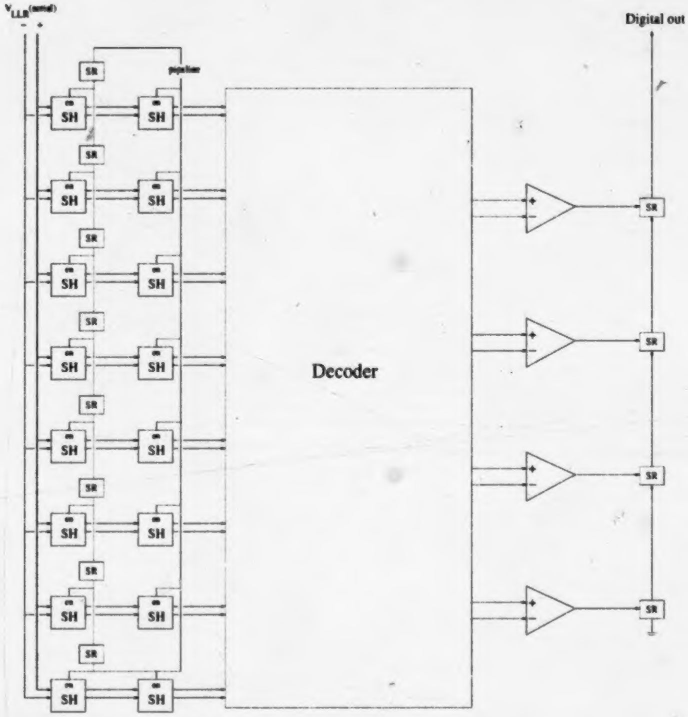


Figure 8.2.1: Interface diagram for the (8,4) analog decoder.

8.2 Interfaces.

To perform serial-to-parallel conversion of analog samples, an array of differential S/H circuits is used. These circuits sample analog data from a common analog bus, as indicated in Figure 8.2.1. The timing of the S/H circuits is controlled by a chain of shift registers (SRs). The "enable" signal is passed from one shift-register to the next, until a full codeword has been received. An array of latching current comparators is used at the output.

A clock-generator circuit coordinates the comparator latch signal and the input select signals for the S/H buffer. There are eight separate select signals, as indicated in the timing diagram shown in Figure 8.2.2. Each select signal is enabled, then disabled, sequentially until a block of samples is received. A global "reset" input (provided from off-chip) signals the start of a block.

A "pipeline" signal coincides with the eighth select signal, and causes all stored

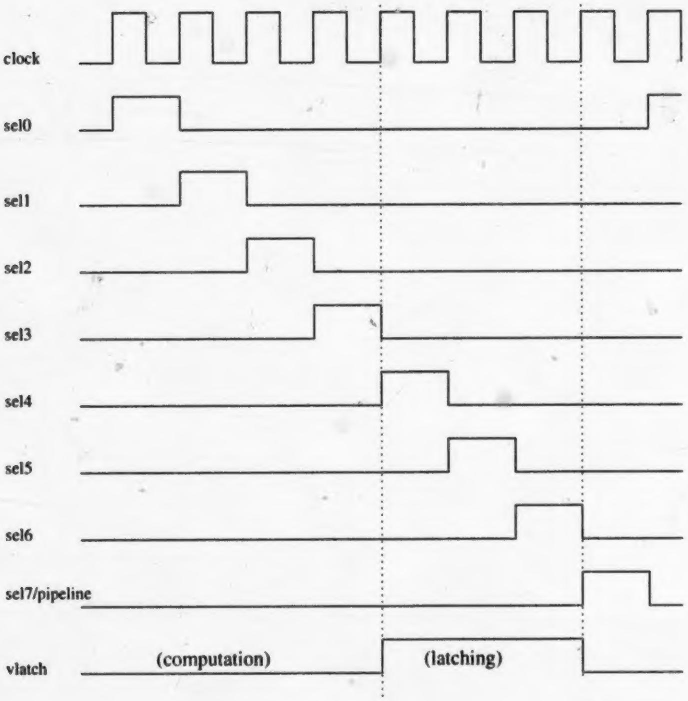

clock
sel0
sel1
sel2
sel3
sel4
sel5
sel6
sel7/pipeline
vlatch
(computation)
(latching)

Figure 8.2.2: Timing diagram for the analog (8,4) Hamming decoder interface.

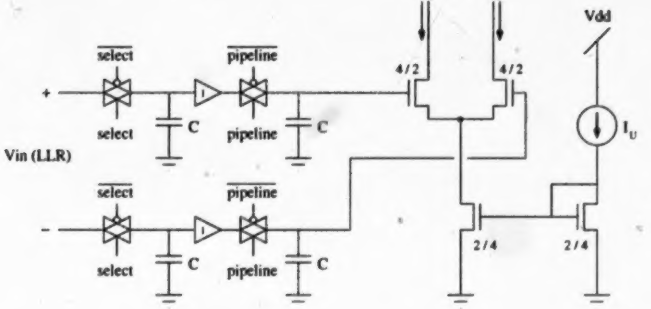


Figure 8.2.3: The input stage for the (8,4) decoder.

samples to be re-sampled simultaneously by a second S/H buffer. The second buffer holds the samples, presenting them in parallel to the decoder until decoding is complete. Five clock cycles are allocated for decoding (including the time during which "pipeline" is high). During decoding, the comparator latch signal, "vlatch," is low. The "vlatch" signal is high during the fifth through the seventh clock cycles. This pipelining scheme results in a two-codeword delay before outputs can be sampled.

8.2.1 S/H input circuits.

The input stage for the (8,4) analog decoder is shown in Figure 8.2.3. The Width/Length of each transistor are indicated as 4/2 and 2/4, where the units are μm. The N and P type transistors in the transmission gates have size $W/L = 1.8\mu m/0.6\mu m$.

The two S/H stages are isolated by a unity-gain buffer, shown in Figure 8.2.4. The buffer has a -3dB frequency of 100MHz. Each S/H sub-circuit uses a 200fF capacitor. The S/H circuits were fully characterized and optimized by Shuhuan Yu, who also provided the buffer design [96].

8.2.2 Comparator circuit.

The final bit decisions are made by a latched current comparator, shown in Figure 8.2.5. Monte Carlo SPICE simulations were performed by Shuhuan Yu on the comparator circuit, using a measured estimate of mismatch characteristics for the AMI process [96]. Based on this analysis, the input offset of the current-comparator has

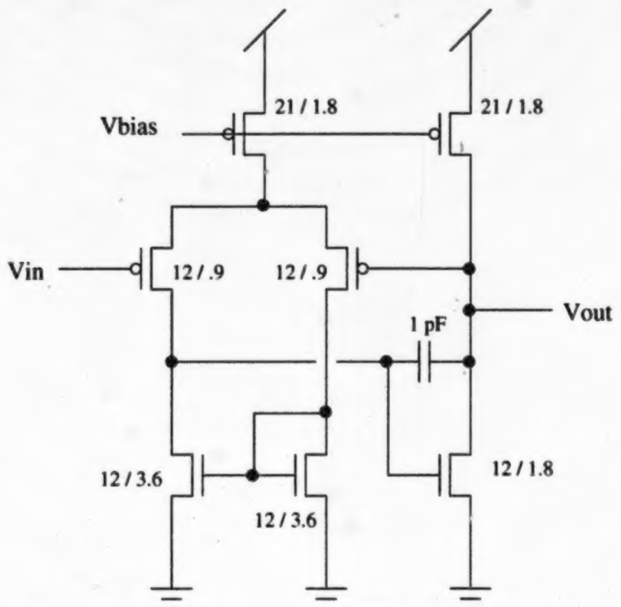


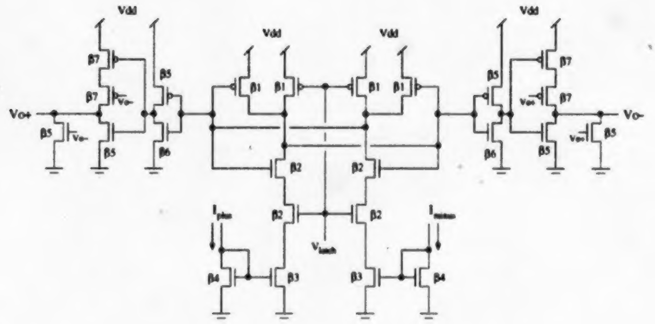
Figure 8.2.4: The unity-gain buffer circuit.

an estimated standard deviation of between 15% and 20% of the operating current I_U.

8.3 Physical test results.

The decoder chip, shown in Figure 8.3.1, was fabricated in an AMI 0.5μm process. A second chip containing test structures was also fabricated. Basic design features of the decoder chip are summarized in Table 8.1. Transistor sizes are reported for the core decoder circuit, in which each transistor has a W/L ratio of 2 for transistors used in Gilbert multipliers, and 0.5 for transistors used in current mirrors. The reported decoder power consumption refers to the power consumed in the core decoder, excluding the interfaces. The chip's behavior was verified at speeds from 1kbps to 2Mbps. Typical power consumption is between 10 and 100 μW, corresponding to speeds between 1 and 100 kbps.

Using a pair of arbitrary waveform generators to produce input samples and a synchronized clock signal, the chip is tested in full-speed serial mode. A Matlab


Vdd
Vdd
Vdd
Vdd
Vo+
Vo-
Vo-
Vo-
Vo+
Vo+
β1
β2
β3
β4
β5
β6
β7
I_plus
I_minus
V_latch

Transistor sizes (W/L):
$\beta 1 = 1.8\mu / .9\mu$
$\beta 2 = 3\mu / .9\mu$
$\beta 3 = 5.1\mu / 2.1\mu$
$\beta 4 = 3.9\mu / 2.1\mu$
$\beta 5 = 3\mu / .6\mu$
$\beta 6 = 1.8\mu / .6\mu$
$\beta 7 = 6\mu / .6\mu$

Figure 8.2.5: Latched current comparator circuit.

Table 8.1: Summary of (8,4) Hamming decoder characteristics.

Die Size	1.5mm×1.5mm
Technology	0.5μm 3M 3.3V CMOS
Circuit Area	0.81mm^2
Decoder Area	0.08[illegible]mm^2
Transistor Size	2μm × 4μm
Tested Speed	up to 2Mbps
Core Decoder Power	1mW at 1Mbps (I_U =2μA) 16μW at 20kbps (I_U = 58nA)
Digital Power	44.2mW
Pad Power	135μW

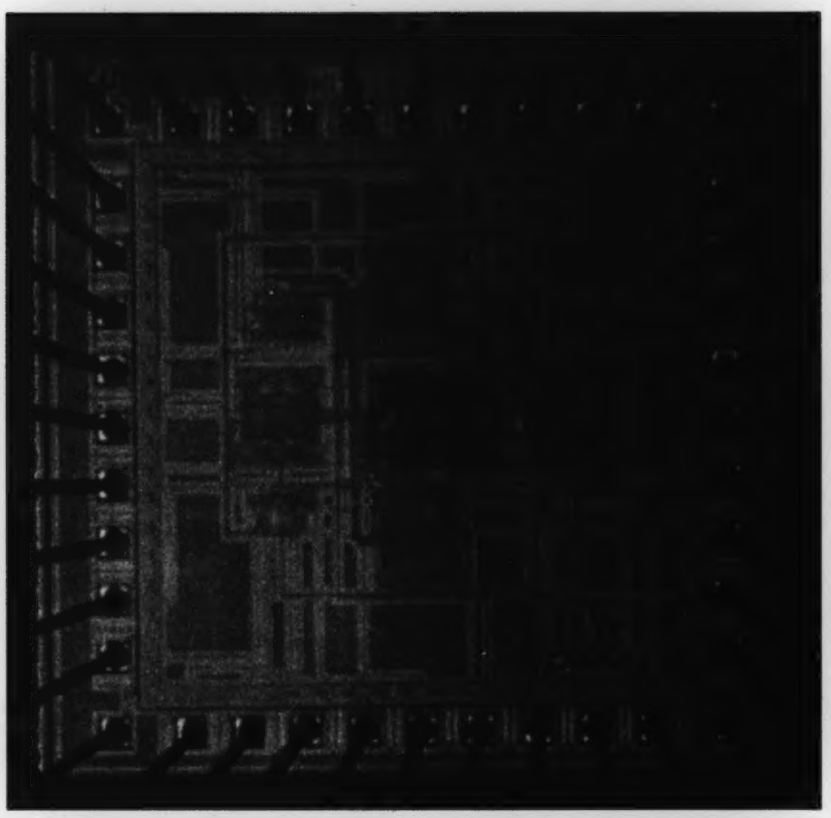


Figure 8.3.1: Photo of the analog (8,4) decoder chip.

script is used to generate random information bits and encode them. The script then adds Gaussian noise samples to simulate the Additive White Gaussian Noise (AWGN) channel. The resulting samples are appropriately scaled so that they represent LLR values, thereby simulating the output of an idealized matched-filter receiver. The LLR samples are sent to the waveform generator via a GPIB interface, where they are provided to the chip as a serial input stream. The chip's digital outputs are sampled by an oscilloscope and returned to the Matlab script, which counts the errors.

8.3.1 Dynamics of the decoder.

The decoder's maximum throughput (the number of decoded bits per second) depends on the bias current I_U. SPICE simulations give an indication of the allowable operating speed, based on the cross-over time and the 90% rise time of the analog output decisions.

The cross-over time is the instant at which the *sign* of a differential output has attained its final value. The rise time is the time it takes for the output to attain 90% of its final *magnitude*. Because the cross-over and rise times may vary from sample to sample, we may only use them to roughly estimate the maximum speed. The chip should operate somewhere between the limits predicted by the rise and cross-over times.

With our test setup, the variation of performance with speed is most conveniently observed when the decoder's speed is limited to between 1kHz and 10kHz. At $I_U = 58\,nA$, the 90% rise-time for a transition in the output decisions is about $150\mu s$, corresponding to a speed of $27\,kbps$. The output cross-over time (the time after which the actual decision changes) is about $90\mu s$, for a speed of $44\,kbps$, which should give some indication of the maximum possible speed. Performance results for this I_U at different speeds are shown in Figure 8.3.2. The power consumed in the core decoder at this speed is $16\mu W$.

All reported bit error-rate measurements have a 95% confidence interval of better than ±30%. Results are not available for the fourth output bit. A simple parallel/serial output mode-select circuit suffered from a floating node which should

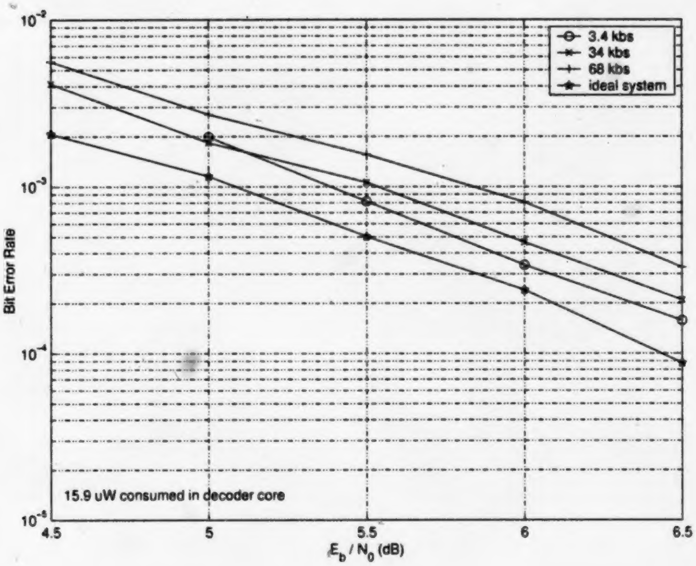


Figure 8.3.2: Decoder performance vs speed in moderate inversion.

have been connected to ground. This mistake results in a stuck output on the fourth bit. The analog outputs of this bit are still measurable, but off-chip current comparators introduced additional problems such as glitches, phase shifts and limited speed, which corrupted test results. The reported results therefore represent the three observable digital outputs.

8.3.2 Measurements in strong inversion.

Due to limitations in the oscilloscope, the serial-mode chip test can only measure performance at speeds above 1 kbps. The code's small block length requires moderate-inversion biasing ($I_U > 14nA$) to achieve testable speeds. While designed to operate in weak inversion, the chip also functions with strong inversion bias currents, and has been tested up to $I_U = 4\mu A$, which is well into strong-inversion.

Some performance loss occurs in strong inversion, as seen in the $I_U = 1.74\mu A$ measurement reported in Figure 8.3.3. The test was conducted at a speed of 424 kbps. The performance loss at this bias current agrees closely with simulations.

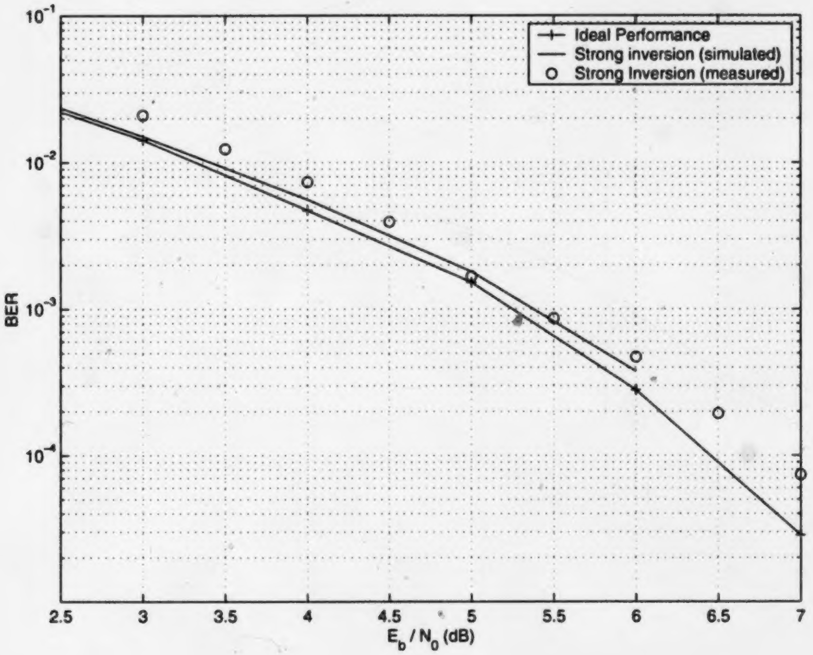


Figure 8.3.3: Measured performance in strong-inversion.

The solid curve in Figure 8.3.3 was obtained using a hybrid analog model implemented in VHDL. The model includes square-law transistor behavior and a one-pole system to model the dynamics of Gilbert multiplier circuits [24]. The close fit between simulated and measured points is taken to be a validation of this simulation model. Figure 8.3.3 reports simulation results alongside performance of an "ideal" Hamming decoder. The distance between the ideal and measured curves is accounted for by moderate-inversion biasing. The performance loss of roughly 0.3dB at BER=7×10^{-5} matches the prediction made by high-level simulations.

The measured points of Figure 8.3.2 represent the performance averaged over the three observable bit positions.

8.3.3 Measurements in weak inversion.

Using a specialized test interface, detailed in Section 9.3, it was possible to retest the (8,4) Hamming decoder operating in weak inversion. The results reveal subtle effects in the decoder's dynamic transition from one sample block to the next.

One set of results, shown in Figure 8.3.4, reveals exactly the expected performance. The Figure shows solid curves representing the minimum-distance asymptote for the (8,4) Hamming code, and the performance of uncoded BPSK. Measured performance from the decoder is indicated by circles. 99.9% confidence intervals are also shown.

The performance in Figure 8.3.4 is obtained by sending the same codeword each time, varying only the noise. The same results are obtained, regardless of which codeword is sent.

If the transmitted codeword is varied randomly, there is a dramatic change in performance. The performance in this case is shown in Figure 8.3.5. The performance loss is attributed to the decoder's "memory" of previously decoded samples. In the design of the (8,4) Hamming decoder, there is no mechanism for clearing the decoder's state before proceeding to a new block of samples. Thus the outcome of one sample block can bias the decoding of subsequent blocks. This problem can be remedied by the inclusion of a reset circuit. Such a reset circuit is incorporated into the designs of Chapters 9 and 10.

8.3.4 Mixed-signal interference.

A measurable amount of interference from on-chip digital circuitry occurs on the first bit position, due to the layout proximity between the analog outputs for those positions and the comparator latch signal. Figure 8.3.6 shows the measured analog decoder output of a pair of output pins with interference. The two waveforms represent the probability values for an output bit.

The analog output wire labeled 'p1' in Figure 8.3.6 was found, upon examination of the layout, to be routed parallel to the "vlatch" signal wire, at minimum spacing, for a distance of 187μm. The discontinuities in the interference pattern correspond precisely to the rising and falling edges of the vlatch signal. Interference from vlatch is visible on one of the other analog outputs, but it is comparatively faint. This amount of interference seems to result in a very small performance loss on the affected bit position, but the precise amount of loss is too small to be resolved

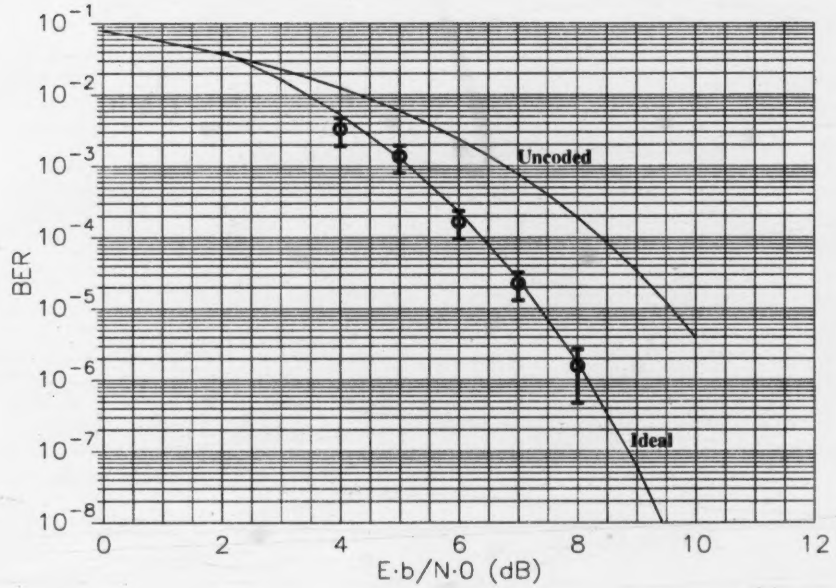


Figure 8.3.4: Measured performance in weak inversion, sending only one codeword. The solid curves represent uncoded BPSK and ideal Hamming code performance. Measured points are indicated by circles. Error bars indicate 99.9% confidence intervals.

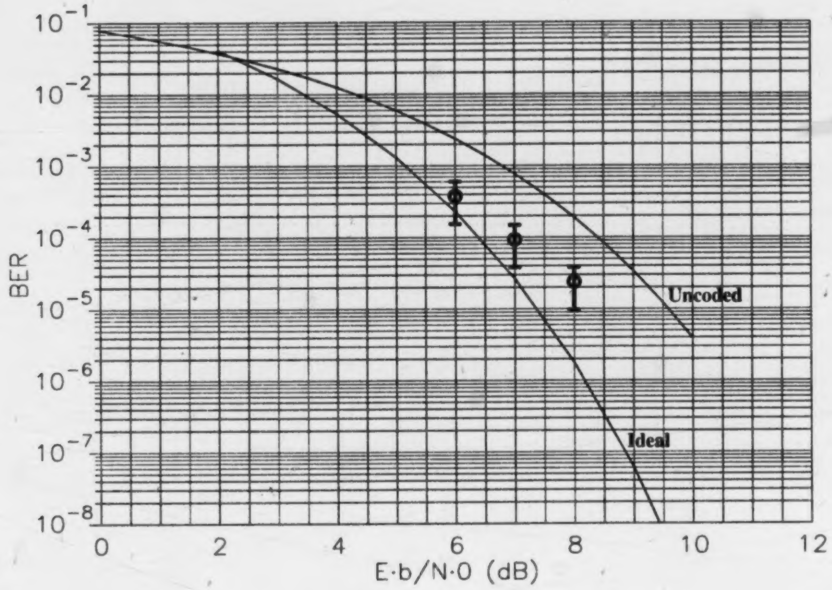


Figure 8.3.5: Measured performance in weak inversion, varying the codeword. The solid curves represent uncoded BPSK and ideal Hamming code performance. Measured points are indicated by circles. Error bars indicate 99.9% confidence intervals.

by the current test method.

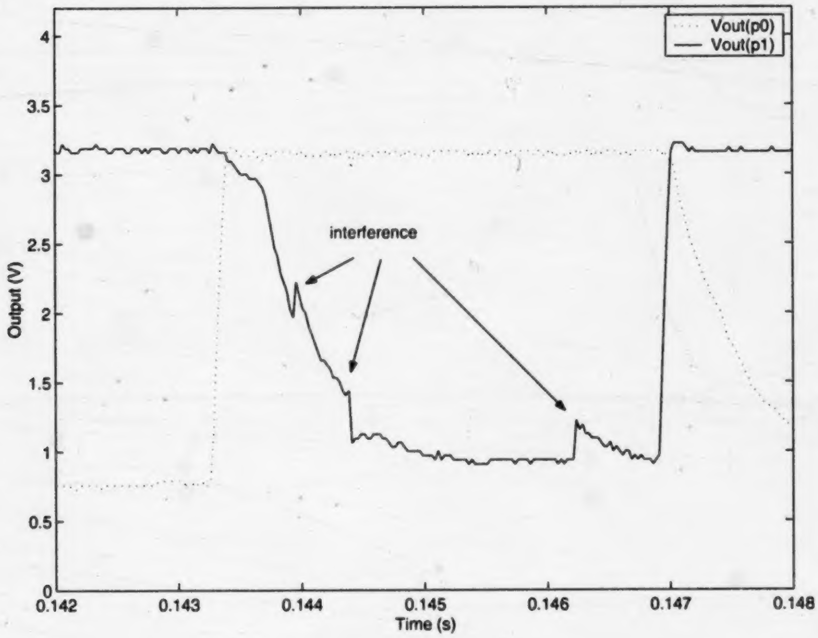


Figure 8.3.6: Mixed-signal interference in outputs of the (8,4) analog decoder.

Chapter 9

*A CMOS Analog Decoder for a (16,11) Hamming Code.

In this chapter we present an analog implementation of the conventional (16,11) Hamming trellis decoder developed using the squaring construction in Section 3.3.4. We again use the canonical CMOS sum-product method of Section 5.2.2. This is also the first fabricated decoder to make use of the reference input method described in Section 5.4.

The decoder was implemented in a TSMC six-metal 0.18μm six metal digital CMOS process. Only the lowest three metal layers are used in the component layout. All transistors in sum-product circuits are sized 1.15μm × 0.35μm (W by L), which is about 1.9 times the minimum transistor length for this process, and five times the minimum width. The circuit accepts serial-mode analog differential voltages corresponding to log-likelihood ratios. The circuit produces serial-mode digital bits at its output.

The S/H interface and latched current comparators of Section 8.2 are also redesigned to make them suitable for the 0.18μm process. A specialized testing board has been constructed, based on an FPGA, a DAC, and a C++ user interface, which interfaces with the analog decoder. In this chapter, we present some details of the test interface, and examine issues in its use.

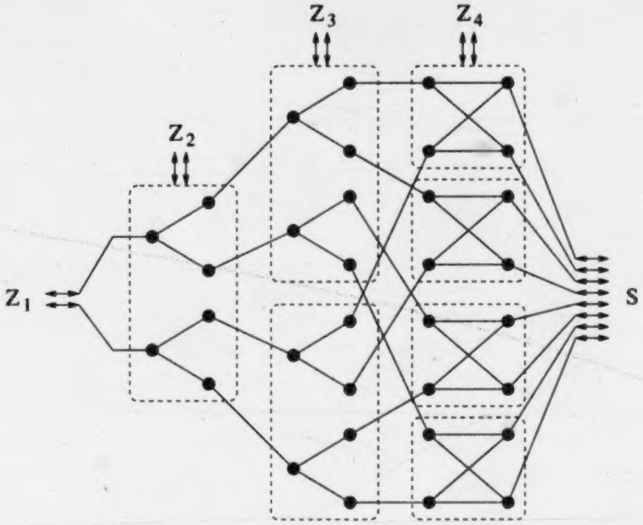


Figure 9.1.1: Subdivision of the bit-combiner (Figure 3.3.10) into atomic trellis components. The labels Z_j represent information input and output for binary code variables. S represents a hidden state of the eight-state trellis graph.

9.1 The analog sum-product components.

The analog components for this decoder are derived from the trellis of Figure 3.3.9. The decoder is divided into two major components: the *bit combiner* and the *trellis core*. The bit combiner itself is divided into three sections. Each trellis section requires three circuits to implement "forward," "backward," and "upward" sum-product computations. Chapters 5 and 8 demonstrate how these circuits may be synthesized from trellis descriptions.

The structure of the bit-combiner trellis allows it to be subdivided into simple "atomic" trellis subsections. This subdivision is illustrated in Figure 9.1.1. The first two stages of the bit combiner can be constructed using only the "tree" trellis structure. In the third bit combiner section, a set of "butterfly" trellis components is used.

When disjoint subsections are used to construct a trellis section, they must sometimes be implemented using a "reference input," as explained in Section 5.4. The reference input is required only when the *row inputs* are divided among disjoint subtrellises. For each of the trellis sections of Figure 9.1.1, three circuit implementations are required. Some of these require reference inputs, others do not.

The core trellis section similarly requires three sum-product circuits. The geometry of the core section fortuitously dictates that all three of these circuits are identical. Moreover, because the core trellis section is divided into identical disjoint subtrellises, the sum-product circuit may be simplified to a repetition of the sum-product circuit for the subtrellis. The design for the (16,11) core subtrellis circuit, using a reference input, is examined in Example 5.4.2.

The complete decoder is therefore constructed using only the three atomic subtrellises shown in Figure 9.1.2. Because of the symmetry of these subtrellises, only six distinct circuit implementations are required:

- *Tree*
  - Forward, backward, and upward computations,
  - Upward computation with reference input,
- *Butterfly*
  - Upward computation with reference input,
- *Core*
  - Forward/backward/upward computation (all isomorphic) with reference input.

The trellis section implementations are placed in a pinwheel configuration, as shown in Figure 9.1.3. The *bit combiner* sections are enclosed by dashed boxes. Each box in the schematic represents a group of local sum-product circuits for the forward, backward, and upward directions. The box labeled "T" represents an instance of the *Core* subsection. The bus widths are also indicated between each module, where a width of "8" implies 8 wires in the forward direction *and* 8 wires in the reverse direction.

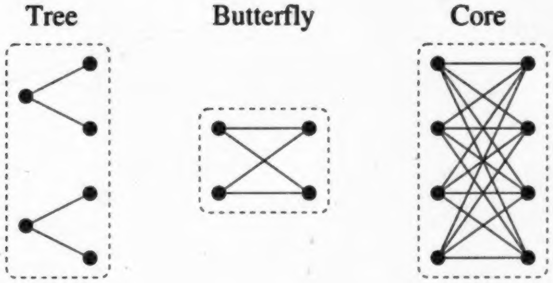

Tree
Butterfly
Core

Figure 9.1.2: Atomic subtrellises for the (16,11) Hamming code.

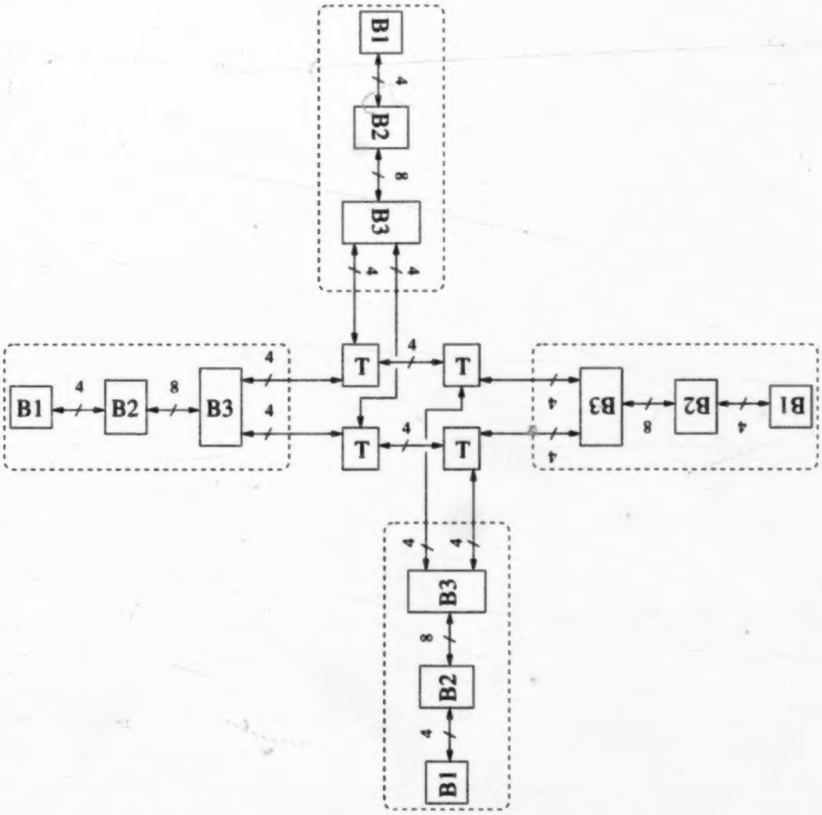

B1
4
B2
8
B3
4
4
T
T
T
T
4
4
B1
4
B2
8
B3
4
4
B3
8
B2
4
B1
4
4
B3
8
B2
4
B1

Figure 9.1.3: A block schematic for the (16,11) decoder.

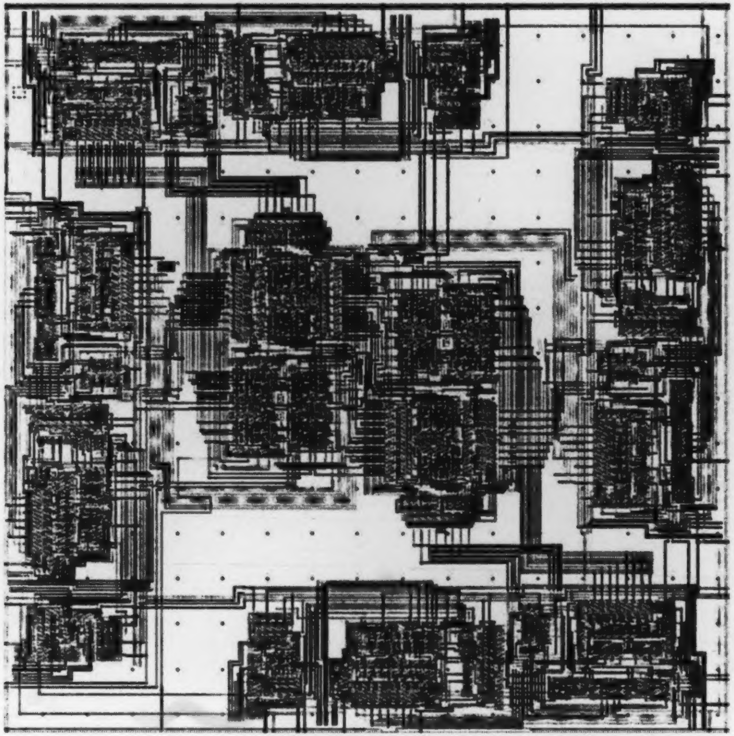


Figure 9.1.4: Layout of the (16,11) analog decoder.

The pinwheel configuration is also visible in the decoder's layout, which is shown in Figure 9.1.4. The layout for the (16,11) decoder is nearly square, and is 163μm on each side.

The implementation presented in Figures 9.1.3 and 9.1.4 uses strictly the sum-product algorithm in accordance with the Extrinsic Information Principle (definition 2.4.1). As a result, the circuit's output for each bit is *extrinsic information* only. Thus the probability mass output for bit u_i is conditioned on every sample in the block except y_i. To complete the computation, the channel information for y_i must be included. This is done through the insertion of an equality node at the output of each bit, as illustrated in Figure 9.1.5.

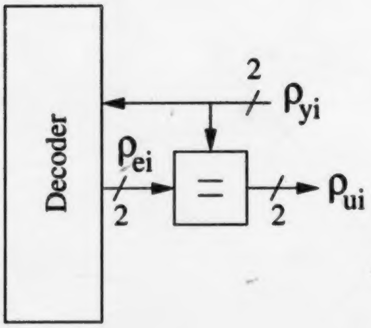

2
ρyi
Decoder
ρei
=
2
2
ρui

Figure 9.1.5: Use of equality gates at decoder outputs.

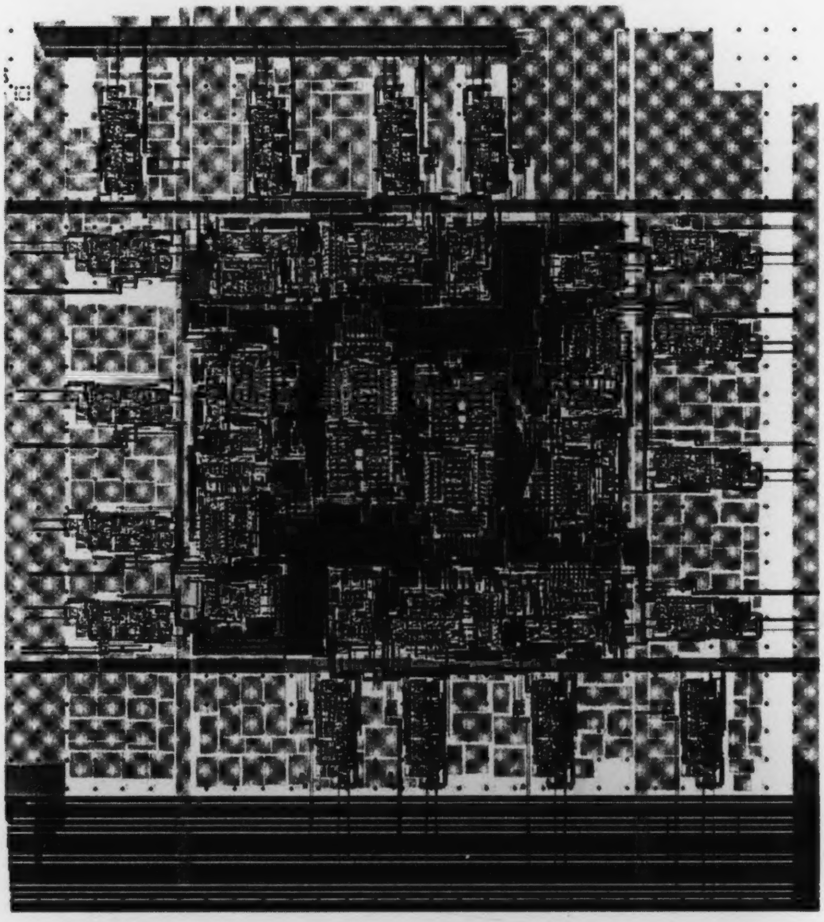


Figure 9.1.6: Layout of (16,11) decoder, with equality gates.

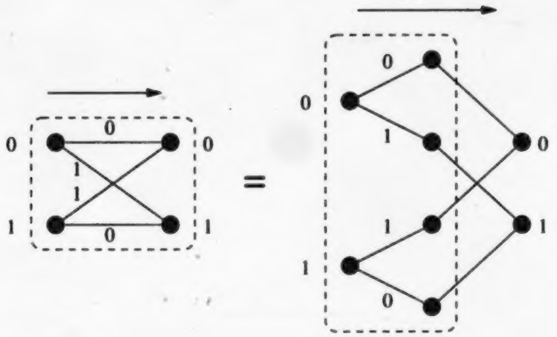


Figure 9.1.7: Creating a butterfly from a tree.

9.1.1 Subtrellis implementations.

The *Tree* circuit is rudimentary, and was examined in Chapter 8. The *Butterfly* circuits in the forward and backward direction are equivalent to a *Tree* circuit, varying only the connectivity of the output nodes. This is illustrated in Figure 9.1.7. The forward and backward *Butterfly* computations are therefore implemented by *Tree* circuits.

The *upward Butterfly* computation requires use of a reference input, as illustrated in Section 5.4.2. The circuit for the *upward Butterfly* computation is shown in Figure 5.4.5. The *Core* circuit was developed in Example 5.4.2. The equality node circuit was also studied in Chapter 5, and is shown on the left-hand side of Figure 5.2.7.

The only remaining sum-product circuit used in the (16,11) design is the *upward Tree* computation. For completeness, this circuit is shown in Figure 9.1.8.

9.1.2 Reset switches.

It is desirable to reset the decoder to a uniform state after each decoded block of samples, before proceeding to the next block. A simple method for erasing the decoder's "memory" is the use of reset switches, which are simple pass transistors [53]. The reset switches are used to short all of the wires in a probability mass. This equalizes their node voltages and creates a uniform distribution.

In the (16,11) circuit, reset switches are used in the *Core* trellis component.

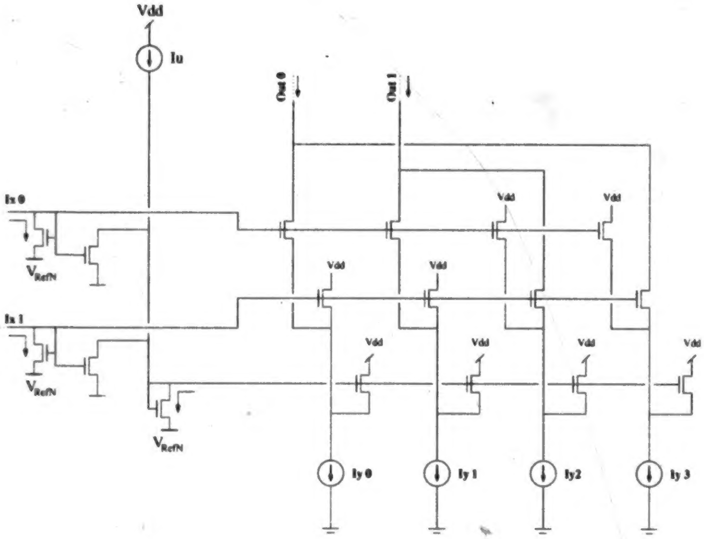


Figure 9.1.8: Upward Butterfly computation, with reference input.

Their connections are indicated in Figure 9.1.9.

9.2 Interfaces.

The interface design for the (16,11) decoder is nearly identical to that of the (8,4) decoder presented in Section 8.2. The (16,11) interface is a generalization of Figure 8.2.1, and is divided into eight-sample subarrays which can be connected end-to-end to create an input interface of arbitrary size.

Within a subarray, a series connected sequence of shift-registers is used to pass control from one S/H circuit to the next. This arrangement is illustrated in Figure 9.2.1. The "top" and "bottom" modules differ only in their handling of control signals.

The "top" module receives a global "frame reset" signal, which synchronizes the interface with the first sample of an incoming codeword. The reset signal is forward from the top module to other modules in the chain. During synchronization, the contents of each shift-register (SR) module are set to zero, except for the first SR which is set to one.

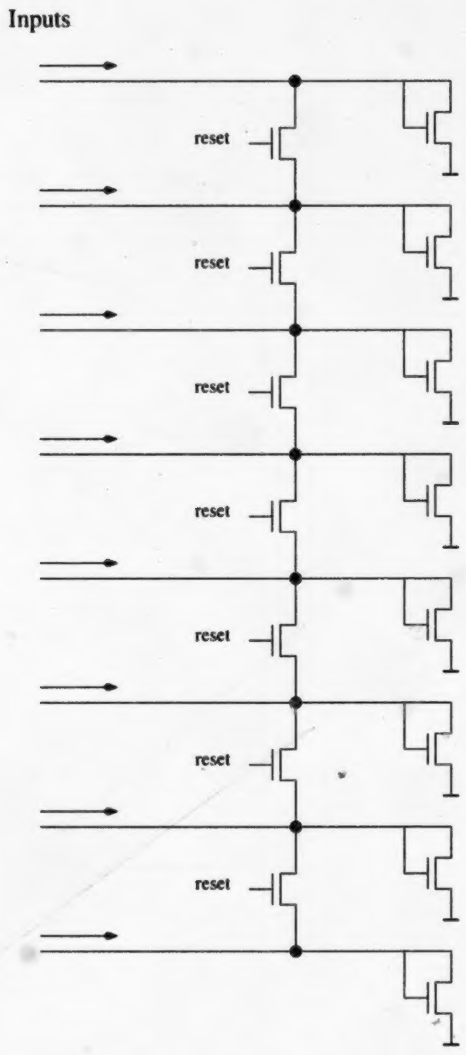

Inputs
reset
reset
reset
reset
reset
reset
reset

Figure 9.1.9: Use of reset switches in the *Core* sum-product component.

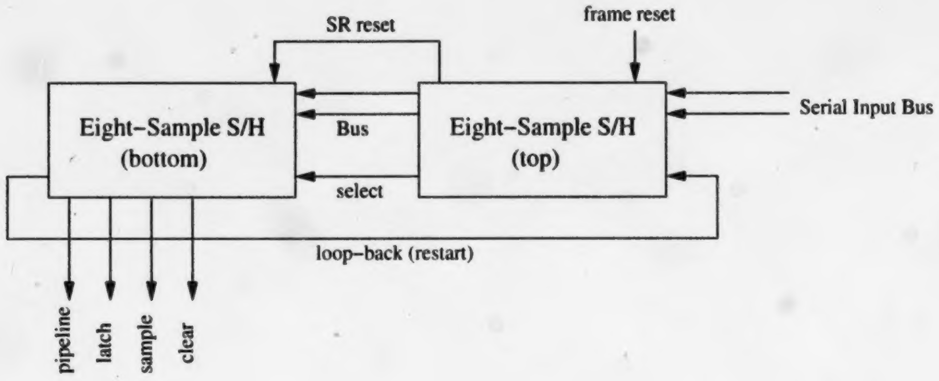


Figure 9.2.1: Modular S/H array.

The "bottom" S/H module generates timing signals for the rest of the system. The "pipeline" signal causes samples to be moved into a second stage of S/H cells. The "latch" signal causes the comparators to resolve their final bit-decisions. The "sample" signal causes the out SR array to latch the comparators' outputs. The "clear" signal causes the second stage of S/H cells to empty their charges just prior to the "pipeline" event.

The signal timings for the "bottom" S/H module are shown in Figure 9.2.2. The labels "sel3," "sel4," and so on refer to the numerical indices of S/H cells local to the module. Thus if the block length is sixteen, "sel3" actually refers to the eleventh sample time.

The comparator's input nodes are allowed to float until "latch" is activated. As indicated by dotted vertical bars in the timing diagram, the comparators are given two clock periods to rail to their final decisions before the "sample" event. Immediately after sampling, the "clear" signal is activated.

An extra clock is devoted to the "pipeline" signal before proceeding to the next block of samples. This means that for a block length of n, the input interface requires $n+1$ clocks to latch in a complete block of samples. In the (8,4) analog interface, "pipeline" coincided with "sel8." This was possible because the (8,4) interface employed a unity-gain buffer between S/H stages. No such buffer is used in the (16,11) design, so an extra clock is required.

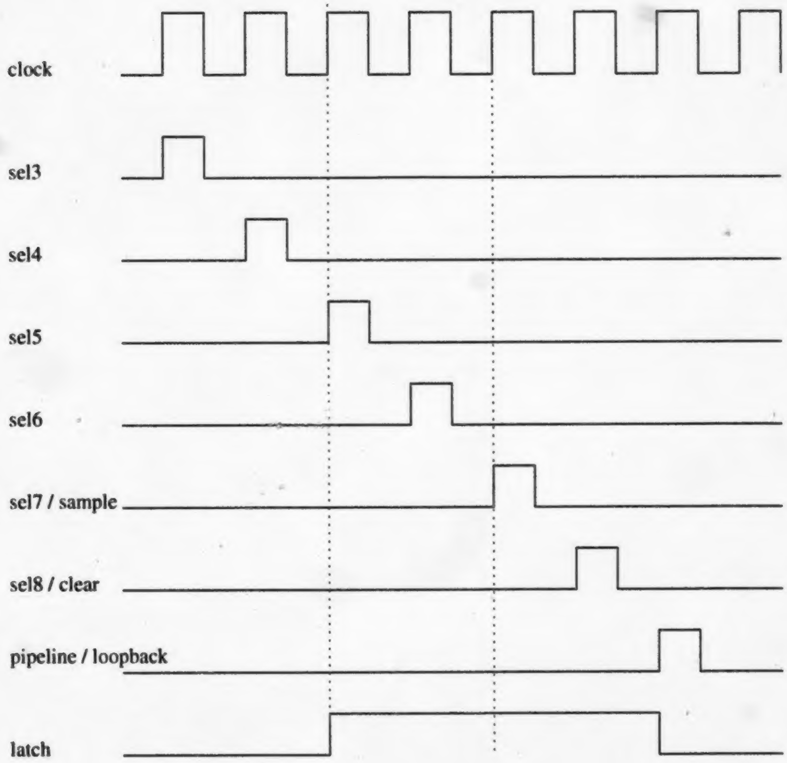

clock
sel3
sel4
sel5
sel6
sel7 / sample
sel8 / clear
pipeline / loopback
latch

Figure 9.2.2: Signal timings for the bottom S/H array module.

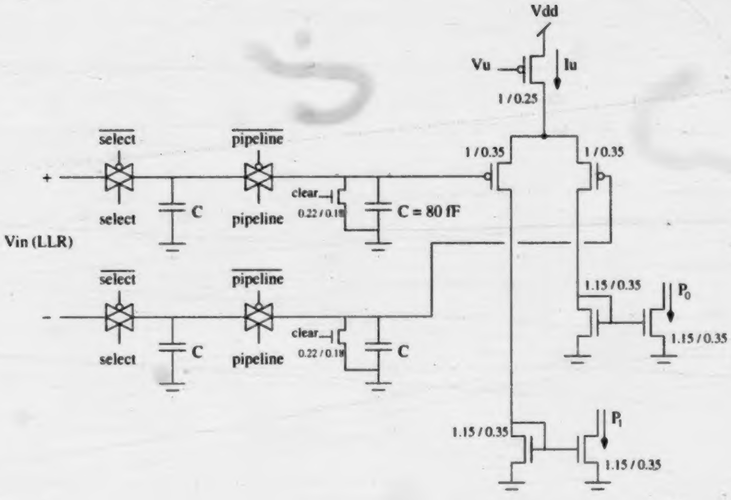

Vdd
Vu
Iu
1 / 0.25
$\overline{\text{select}}$
$\overline{\text{pipeline}}$
1 / 0.35
1 / 0.35
+
select
C
clear
pipeline
0.22 / 0.18
C = 80 fF
Vin (LLR)
$\overline{\text{select}}$
$\overline{\text{pipeline}}$
1.15 / 0.35
P_0
-
1.15 / 0.35
clear
select
C
pipeline
0.22 / 0.18
C
1.15 / 0.35
P_1
1.15 / 0.35

Figure 9.2.3: S/H circuit for the (16,11) decoder.

9.2.1 S/H input circuits.

The input S/H circuit for the serial input module is shown in Figure 9.2.3. There is no buffer between the S/H stages, so all voltages are divided in half when they reach the second stage. To accommodate a lower common-mode voltage, P-type differential pairs are used. Common centroid layouts are used for the differential pairs to minimize mismatch.

The transmission gate, detailed in Figure 9.2.4, utilizes a well-known techniques for reducing charge-injection. A pair of dummy transistors, P2 and N2, are attached to the output. The source and drain of these dummy transistors are shorted together, so they have no logical effect on the circuit. When P1 switches off, P2 is switched on. The width of P2 is 1/2 that of P1, so that the charge ejected from P1 is mostly absorbed into P2. The same approach applies to the NMOS devices.

9.2.2 Comparator circuit.

As in the (8,4) design, the final bit decisions are made by a latched current comparator, shown in Figure 9.2.5. This comparator employs the same circuit topology as Figure 9.2.5, but with sizes adjusted for the TSMC process.

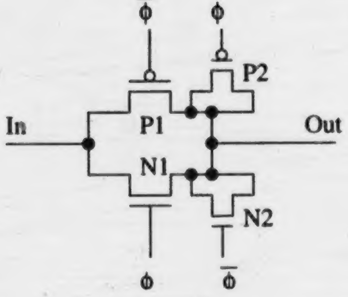
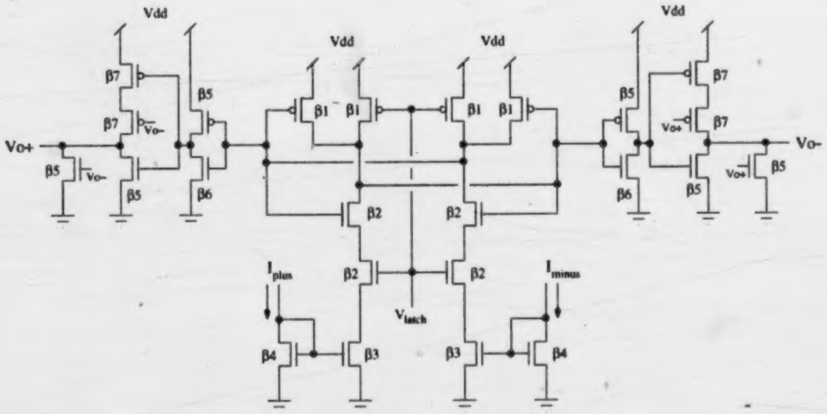

$\bar{\phi}$
ϕ
P2
In
P1
Out
N1
N2
ϕ
$\bar{\phi}$

Figure 9.2.4: Transmission gate circuit.


Vdd
Vdd
Vdd
Vdd
β7
β7
Vo-
β5
Vo+
β5
Vo-
β5
β6
β1
β1
β1
β1
β2
β2
β2
β2
I_{plus}
I_{minus}
V_{latch}
β4
β3
β3
β4
β5
β7
Vo+
β7
Vo-
β5
Vo+
β5
β6
β5

Transistor sizes (W/L):

$\beta 1 = 1.8\mu / .9\mu$

$\beta 2 = 3\mu / .9\mu$

$\beta 3 = 5.1\mu / 2.1\mu$

$\beta 4 = 3.9\mu / 2.1\mu$

$\beta 5 = 3\mu / .6\mu$

$\beta 6 = 1.8\mu / .6\mu$

$\beta 7 = 6\mu / .6\mu$

Figure 9.2.5: Latched current comparator circuit.

9.3 Test interface.

An FPGA test interface was designed and constructed to provide an efficient bridge between the chip's serial analog interface and a desktop computer. In effect, the test interface provides a low-level USB-controlled single channel arbitrary waveform generator, with a synchronized logic analyzer for up to four signals from the device-under-test (DUT). The test interface can also generate synchronized digital control signals (e.g. clock and frame). The test interface work was carried out in partnership with Dave Nguyen.

9.3.1 Hardware.

The test interface is based on the Digilent Digilab 2E FPGA development board, which uses a Xilinx Spartan 2E FPGA device. The Digilab board has a set of board-to-board headers which allow easy communication between the FPGA and add-on boards.

A specialized daughter board was designed and fabricated to mate with the Digilab FPGA platform. The daughter board includes a Texas Instruments AD9764 digital-to-analog converter, which has 14-bit resolution and operates at up to 125 MS/sec. The DAC outputs are differential, and are passed through a high-speed AD8138 analog buffer which allows manual control of the peak-to-peak amplitude and common-mode voltage, which are adjusted by potentiometers on the board.

Also on the daughter board is a set of four potentiometers for setting arbitrary bias voltages. These voltages are buffered through an LM324 operational amplifier array. An adjustable voltage supply is provided by an LM317 voltage regulator. Digital signals are also buffered using LV125 and LVC244A digital driver chips.

To facilitate communication with a desktop computer, the daughter board also includes a DLP245M USB-to-FIFO converter chip. This chip provides a seamless interface between the FPGA and the USB bus. From the FPGA perspective, the USB is simply an asynchronous eight-bit FIFO with a rudimentary handshaking protocol.

Each chip to be tested is placed on its own DUT board, which mates to the

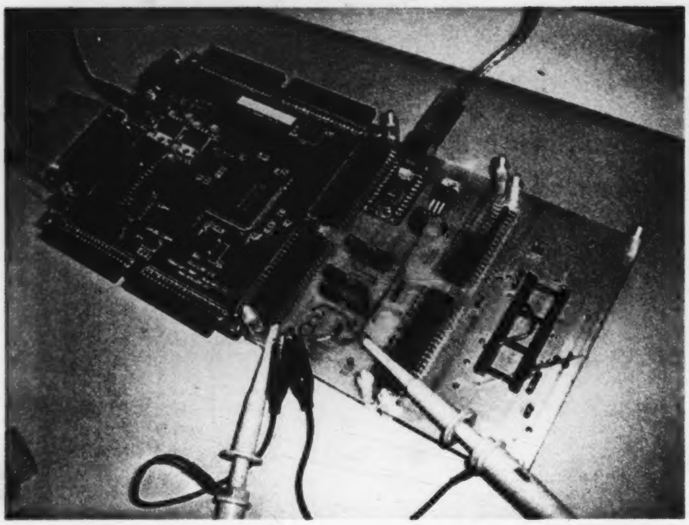


Figure 9.3.1: Photograph of the FPGA-based test board.

daughter board through a second set of board-to-board headers. When analog decoder chips are designed using the generalized serial interface of Figure 9.2.1, this test configuration can be used for any analog decoder, without need for modification. A photograph of the test hardware is shown in Figure 9.3.1.

9.3.2 Software.

To control the FPGA test device, a C++ class was written which allows direct streaming of bytes to and from the USB interface. A set of simulation classes, written using GNU C++ on a RedHat 9 Linux platform, handles the generation of coded bits and Gaussian noise samples, and keeps track of errors.

The software interface allows real-time graphical display of error curves, and produces plots of detailed bit-by-bit information about the decoder's behavior. The graphical interface uses the PlPlot library for scientific visualization.

The test software allows direct interfacing of physical components with sophisticated C and C++ based simulations of communication systems. The real-time display of data and the accessibility of low-level hardware enable rapid discovery of faults, heuristic exploration of DUT behavior, and simple verification of hard-

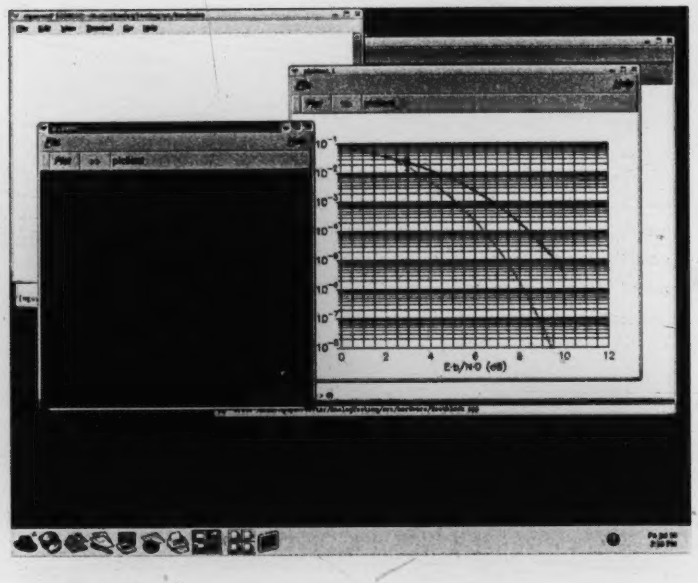


Figure 9.3.2: Screen shot of the test interface during a test of an uncoded loop-back interface chip.

ware performance.

9.3.3 Noise in the test interface.

The test interface introduces its own systematic noise, with variance σ_t^2, as indicated in the interface diagram of Figure 9.3.3. The systematic noise can be measured after sample generation and scaling. If the correct scaling factor is used, namely $s = U_t/\kappa$, then the systematic noise component appears, relative to the decoder, to be additional noise in the LLR value.

To refer the systematic noise to the channel, we therefore ignore s and scale the standard deviation, σ_t, by the factor $N_0/4$. The total channel noise is therefore

$$\begin{aligned}
\sigma_{\text{tot}}^2 &= \sigma_n^2 + \left(\frac{N_0}{4}\right)^2 \sigma_t^2 \\
&= \sigma_n^2 + \left(\frac{\sigma_n^2}{2}\right)^2 \sigma_t^2 \\
&= \sigma_n^2 \left(1 + \frac{\sigma_n^2 \sigma_t^2}{2}\right). \qquad (9.3.1)
\end{aligned}$$

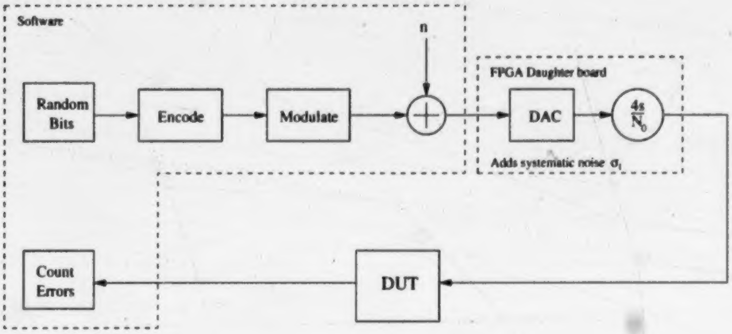


Figure 9.3.3: Diagram of the test interface.

From (9.3.1) we derive the equivalent loss in SNR,

$$\text{loss (dB)} = 10 \cdot \log_{10}\left(1 + \frac{N_0}{4}\sigma_t^2\right). \tag{9.3.2}$$

Because the added noise is independent, additive, white, and Gaussian, it is appropriate to shift the SNR used in sample generation by the amount determined in (9.3.2). Therefore, to measure performance at a given SNR, we physically measure σ_t^2, and then generate samples by assuming $E_b/N_0 = (\text{SNR} + \text{loss}(\sigma_t))$.

9.3.4 Loop-back interface test.

A fully-integrated stand-alone I/O loop chip, in which outputs from the S/H array arrive directly at an array of comparators, was implemented in the TSMC 0.18μm process by Dave Nguyen. The I/O loop circuits are precisely those described in this section.

It is found that the S/H array functions as expected, whereas the comparator design is severely prone to faults. In one tested I/O loop chip, only two of eight comparators functioned with acceptable reliability. For use with practical decoders in a real communication system, a superior comparator design is necessary.

In any given chip, at least one of the comparators is expected to function with a tolerable offset. This is fortunately the case with the I/O chip. The test configuration is validated by measuring the bit error-rate at the output of a good comparator. The results for such tests are shown in Figures 9.3.4 and 9.3.5.

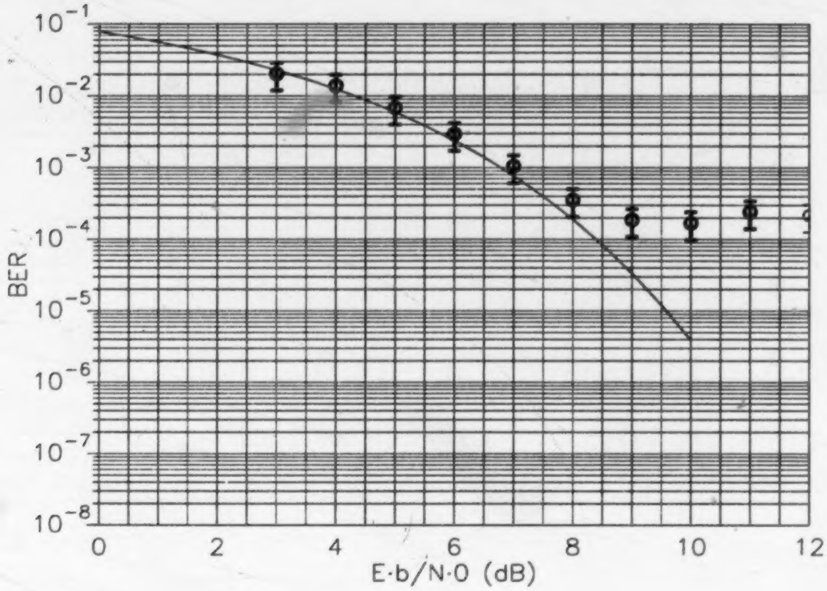


Figure 9.3.4: Results for the loop-back test at maximum speed. The solid curve indicates the ideal performance of uncoded BPSK. Circles indicate measured error rates. Error bars indicate 99.9% confidence intervals.

The results of Figure 9.3.4 were obtained with the test interface operating at maximum speed, which is approximately 20 Msamples per second (MSps). At this speed, high noise and other effects, such as clock feed-through and ringing, were observed at the output of the DAC. This evidently results in a maximum E_s/N_0 of 9dB, as indicated by the error floor in Figure 9.3.4.

The systematic noise is reduced considerably when the sample rate is lowered. The results of Figure 9.3.5 were collected at a sample rate of 2.5 MSps. The noise can also be somewhat compensated in software by increasing the DAC scale, which results in lower resolution for small log-likelihood samples. It is clear that the systematic noise in the system may make it difficult to reliably measure performance for E_s/N_0 greater than 9 dB.

9.3.5 Discussion of the interfaces.

The failure rate of the comparator design of Figure 9.2.5 in no way reflects a fundamental problem for analog decoders. High-quality comparator designs are in

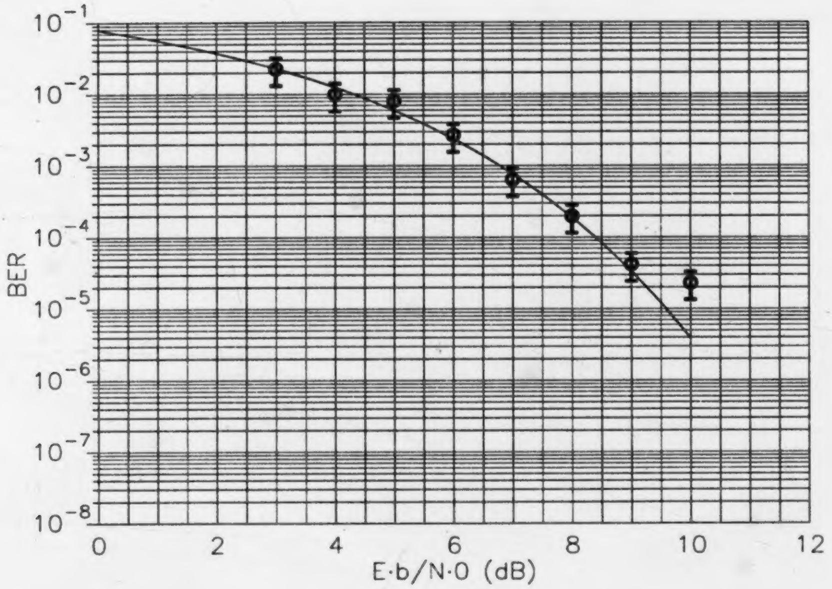


Figure 9.3.5: Loop-back results at reduced speed. The solid curve indicates the ideal performance of uncoded BPSK. Circles indicate measured error rates. Error bars indicate 99.9% confidence intervals.

plentiful supply [8, 93]. The failure is simply due to a poor design choice in the final output interface of the decoder.

Interface problems have plagued analog decoding implementations [54, 90, 86, 44, 53]. This is possibly due to the lack of emphasis which designers place on seemingly trivial, commonplace circuits such as comparators. The central goal of analog decoding research, to date, is to prove the concept. Most of the design effort is directed at the decoding circuits. The interfaces, while of less research value in and of themselves, have perhaps been given inadequate scrutiny.

9.4 Characteristics of the decoder.

The analog (16,11) Hamming decoder was fabricated in a TSMC 0.18μm six-metal digital CMOS process, and some performance tests are complete. The design was implemented on the same chip as the $(16,11)^2$ Product decoder discussed in Chapter 10. A summary of results for the (16,11) decoder design is presented in Table

9.1.

The digital control signals have been verified, and it has been verified that the decoder correctly decodes the all-zero and all-one codewords. A performance curve has also been measured for a subset of bits, and is shown in Figure 9.4.1. The Figure also indicates 99.9% confidence intervals. The confidence interval is wider for the point at 8dB, where only a small number of error observations is available.

To obtain the results of Figure 9.4.1, a subset of bits was chosen to represent the performance of the device. Bit-positions with good performance are readily identified when only the all-zero and all-one codewords are used in the test. In principle, the performance of a decoder can be accurately measured even if the same codeword is transmitted each time, as long as a suitable noise pattern is added to the signal.

With an analog decoder, it is also important to test the "memory" effect, the importance of which is illuminated in Section 8.3.3. When a new block of samples is input to the decoder, the decoder's state is still somewhat influenced by the previous block. The reset switches, explained in Section 9.1.2, are intended to clear the decoder's memory. By alternating between the all-zero and all-one codewords, we detect the effect of any residual memory on performance.

Although most of the output comparators have excessive input offsets, the results of one or a few bits provide significant verification of the decoder's overall performance. Because the decoder performs a block-wise computation, the performance of each bit-position is strongly correlated with that of every other position. Verification of one bit's performance is therefore tantamount to verification of the decoder. If the decoder itself is not properly functioning, then it is impossible for any single bit to show good error performance. We therefore conclude that the decoder's actual functioning is at least as good as the test result in Figure 9.4.1.

9.4.1 Dynamics.

Figure 9.4.2 presents a series of transient simulations of the decoder which can be used to estimate the maximum speed of the device. The outputs shown represent the binary probability mass for one output bit. The operating current for this simulation

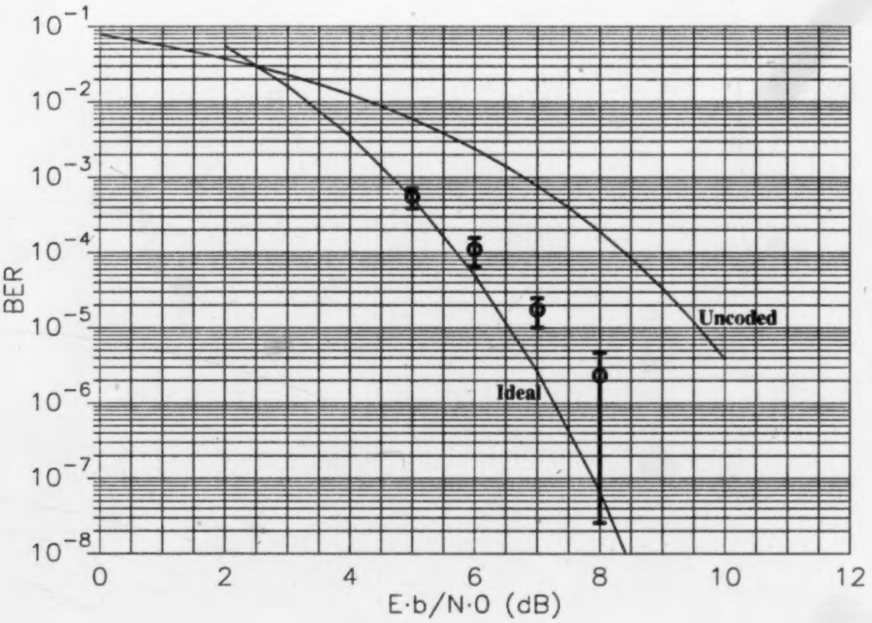


Figure 9.4.1: Performance measurements for the (16,11) decoder. The solid curves indicate performance of uncoded BPSK and an ideal (16,11) Hamming decoder. Circles indicate measured error rates. Error bars indicate 99.9% confidence intervals.

is $I_U = 100\,\mathrm{nA}$.

In Figure 9.4.2, we view the decoder's output in increasingly fine-grained detail. In these views, the decoder's output is seen to respond in an approximately discrete way to signals from its neighbors. Sudden output changes are observed as signals propagate from distant parts of the decoder. In the final view, a minimum response time is visible, which is (to be conservative) roughly $1\,\mu\mathrm{s}$. This corresponds to a maximum clock rate of 1MHz for the overall decoder, resulting in a throughput of 11Mbps.

We may say that the minimum response time is approximately equivalent to a single iteration. A software version of this decoder, using conventional discrete-time iterative decoding, requires about twelve iterations for optimal decoding. Applied to the analog decoder, this gives a thumbnail estimate of $12\,\mu\mathrm{s}$ per codeword (at $I_U = 100\,\mathrm{nA}$). This gives a maximum (uncoded) throughput of 917kbps.

In weak inversion, the device's speed is proportional to its transconductance, which is directly proportional to the bias current, which in sum-product circuits is directly proportional to the unit current, I_U. The power consumed in the decoder is similarly proportional to I_U. By simulating the decoder at different values of the unit current, and estimating the minimum response time, it is possible to express the decoder's speed as a linear function of its power consumption.

We find that the decoder's speed scales at a rate of 50Mbps/mW. For the transistor sizes chosen in this design, and given the parameters of the process, the specific current is in the neighborhood of $10\,\mu\mathrm{A}$. When $I_U = 1\,\mu\mathrm{A}$, the decoder is therefore approaching moderate inversion. Designating this as the maximum unit current, we arrive at an estimated maximum throughput of 135 Mbps for the (16,11) decoder.

The estimated speed applies to the decoder only. The interfaces are less flexible in throughput than the decoder itself. If the decoder is operated too slowly, then the S/H circuits will lose too much of their charge due to substrate leakage, thereby degrading decoder performance. If the decoder is operated too fast, then the S/H circuits may not have time to settle on the correct charge and/or the comparators will not have sufficient time to rail to their correct final decisions.

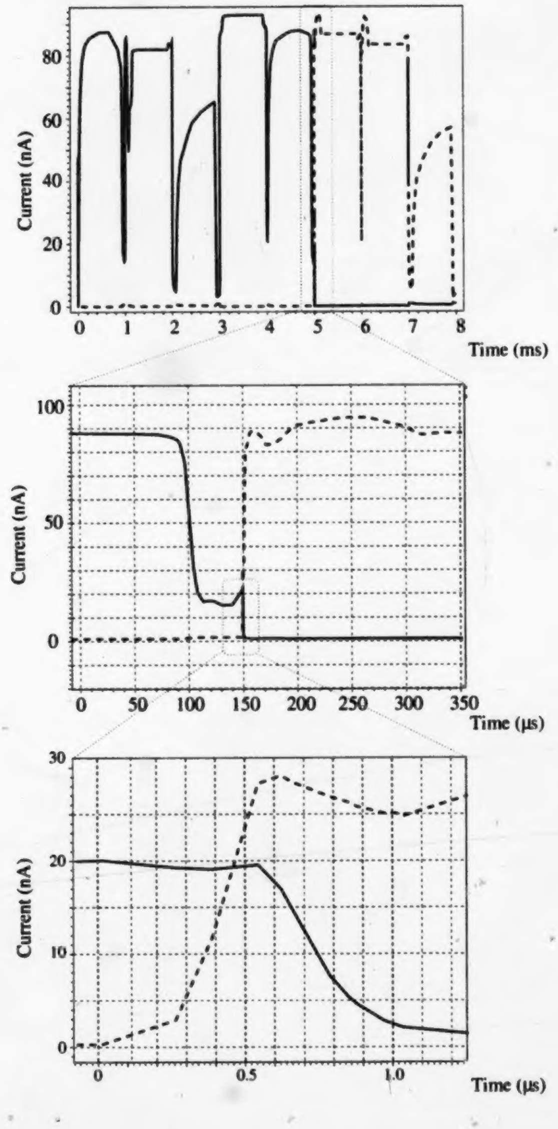


Figure 9.4.2: Transient response of the (16,11) decoder output for a single bit. The solid curve indicates the output $P(\text{bit} = 0)$ and the dashed curve indactes $P(\text{bit} = 1)$.

Table 9.1: Summary of (16,11) Hamming decoder characteristics. Speed and power figures are estimated from simulations, and are not within the range of accurate measurement for the test interface. "Tested Speed" refers to the maximum speed of the test interface.

Die Size	2.3 mm × 2.4 mm
Technology	0.18μm 6M 1.8V CMOS
Circuit Area (with interfaces)	0.24 mm^2
Decoder Area	0.0266 mm^2
Transistor Size	1.15μm×0.35μm
Tested Speed	up to 5 Mbps
Core Decoder Power (est.)	26.9μW @ $I_U = 100$ nA 2.69 mW @ $I_U = 1\,\mu$A
Decoder Speed (est.)	917 kbps @ $I_U = 100$ nA 135 Mbps @ $I_U = 1\,\mu$A
Digital Power	6.84μW
Comparator Power	1.54μW/bit, avg.
S/H Power	1.8μW/bit, avg.
Total Power	87.2μW, avg. @ $I_U = 100$ nA

Chapter 10

*An Analog $(16,11)^2$ Turbo Product Decoder.

In this chapter we present an analog implementation of a $(16,11)^2$ Turbo Product decoder, based on the $(16,11)$ Hamming trellis decoder described in Chapter 9. As of January 2003, the $(16,11)^2$ Product code is part of the IEEE 802.16a standard [3]. A commercial decoder for this code, using conventional digital circuits, is produced by Comtech AHA (Advanced Hardware Architectures) [5].

The Product decoder is constructed by concatenating 32 of the (16,11) component trellis decoders. The resulting decoder has a coded length of 256 bits, making it the largest analog decoder design, as of the writing of this thesis. The largest analog decoder prior to this design is a rate $1/3$ Turbo decoder for the UMTS standard [2], with a coded length of 120 bits [45].

The decoder was implemented in a TSMC 0.18μm 6M digital CMOS process, on the same chip as the single $(16,11)$ decoder of Chapter 9. Connections between component decoders are made using the top three metal layers. As with the $(16,11)$ decoder, the Product decoder circuit accepts serial-mode analog differential voltages corresponding to log-likelihood ratios. The circuit produces serial-mode digital bits at its output. The interface design is precisely that described in Section 9.2. The test interface of Section 9.3 is used to evaluate the behavior and performance of the Product decoder chip.

10.1 Design of the decoder.

The design of the Product code and its decoder are straightforwardly derived from the construction described in Section 2.5.5. A Product codeword has the structure of Figure 2.5.11. Thirty-two instances of the component $(16, 11)$ decoder are used without modification. The component decoders and the I/O interfaces are bridged using equality nodes, as illustrated in Figure 2.5.12.

10.1.1 Floorplan and interleaving.

The component decoders are divided into a set of sixteen *column decoders* and sixteen *row decoders*. Equality nodes are attached to each of the bit positions on column decoders. No equality nodes are attached to the row decoders. A *block interleaving* pattern is then used to interconnect the row and column decoders.

The block interleaving is achieved by arranging the component decoders in a checker-board pattern, as shown in Figure 10.1.1. The pattern can also be seen in the physical layout of the Product decoder, shown in Figure 10.1.2. Note that there is a gap in the lower-right corner of the layout. The single $(16, 11)$ decoder is placed in this location.

Each row and column decoder's input and output wires are arranged in rows and columns, respectively. The interconnect wires, located on the fourth and fifth metal layers, run the full width and height of the Product decoder's layout. Each decoder needs two wires per two directions per sixteen bit positions, thus 64 wires.

10.1.2 Scalability.

The floorplan shown in Figure 10.1.1 is adequate for the present design, but probably would be ill-suited for a larger product code. This is because the bus width of analog decoders tends to grow more rapidly than the decoder itself.

In the present design, the routing for three component decoders must occupy the width or height of a single decoder. For example, the top row of decoders in Figure 10.1.1 includes three row decoders. The routing for all three of these decoder must fit within the height of the row, or the rows will have to be spaced further apart,

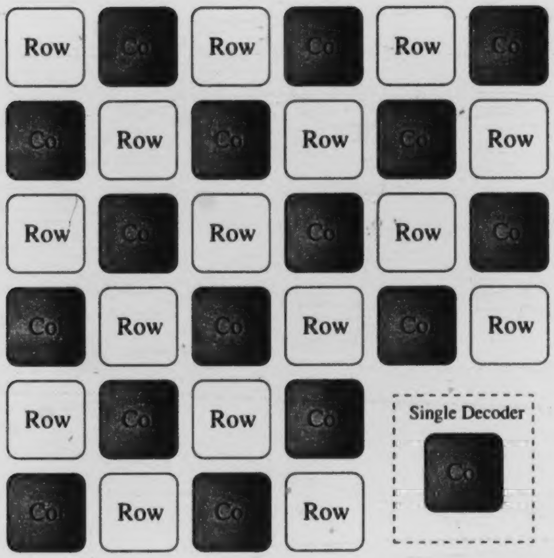

Row
Co
Row
Co
Row
Co
Co
Row
Co
Row
Co
Row
Row
Co
Row
Co
Row
Co
Co
Row
Co
Row
Co
Row
Row
Co
Row
Co
Single Decoder
Co
Co
Row
Co
Row

Figure 10.1.1: Floorplan of the Product decoder.

which is a waste of silicon area.

The routing for three component decoders requires 192 wires. These must be spaced far enough apart to allow connections between every row and column. The resulting height of a bundle of routing wires is slightly greater than the height of a component decoder. Some space is therefore wasted in the layout.

At the outset of the design, the component decoder was expected to be somewhat larger than the final layout, so that no space would be wasted. It is clear that for larger designs, this kind of interleaving is not efficient. A Turbo-like arrangement is no doubt superior for larger designs, as in Figure 10.1.3.

10.2 Characteristics of the decoder.

The analog Product decoder was fabricated in a TSMC 0.18μm six metal process, on the same chip as the (16,11) Hamming decoder reported in Chapter 9. A summary of results for the Product decoder design is presented in Table 10.1. A die photo of the chip is shown in Figure 10.2.1.

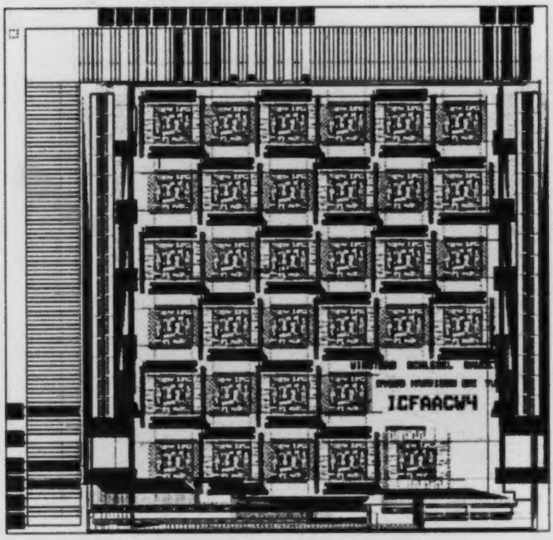


Figure 10.1.2: Layout of the Product decoder chip.

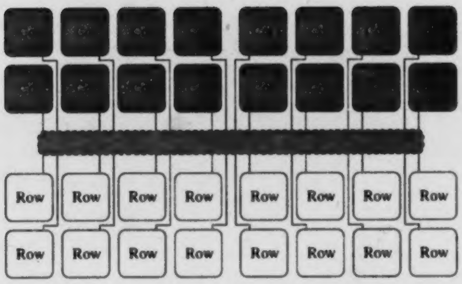

Row
Row
Row
Row
Row
Row
Row
Row
Row
Row
Row
Row
Row
Row
Row
Row

Figure 10.1.3: Alternative Turbo-like interleaver layout.

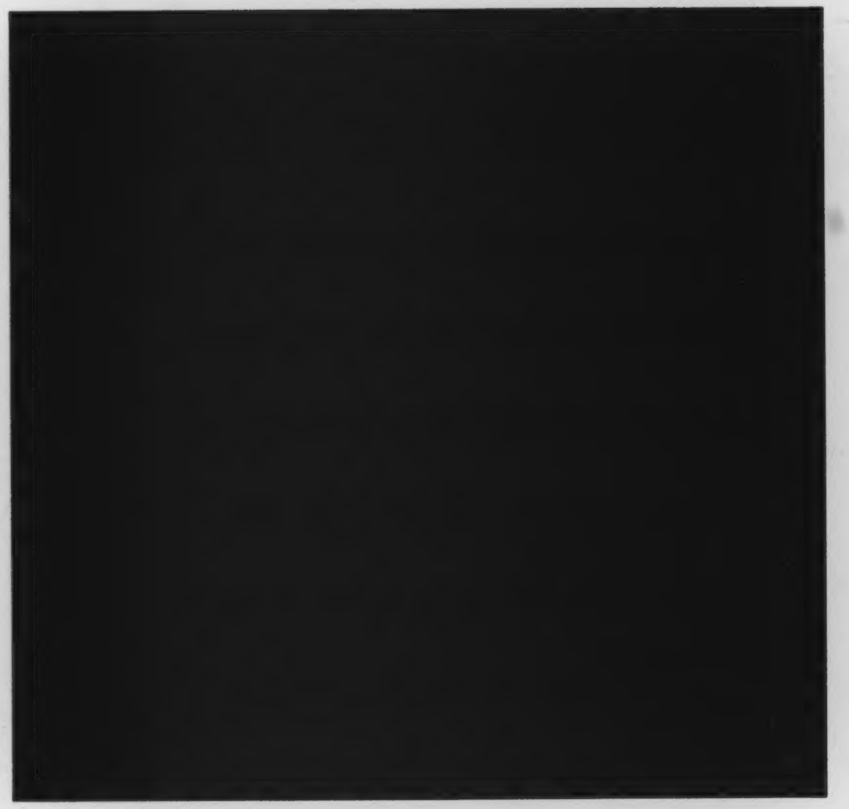


Figure 10.2.1: Die photo of the Product decoder chip.

10.2.1 Speed.

The chip was tested using the interface described in Section 9.3. This interface supports a narrow range of testable operating speeds, from 100 kSps to 3 MSps. Because of its parallelism, the Product decoder can theoretically operate at much greater speeds. Presently untestable characteristics such as maximum speed, and the corresponding power consumption, can be derived from the properties of the $(16, 11)$ component decoder. The component decoder's characteristics are described in Section 9.4. The Product decoder is essentially a repetition of thirty-two identical components.

We use the same estimate of response time for the component decoder as determined in Section 9.4. Assuming that 50 iterations are required for good performance in the product decoder, and noting that the code's uncoded block size is eleven times larger than that of the component decoder, we arrive at the throughput estimates reported in Table 10.1.

The Product decoder's speed scales at a rate of 17 Mbps/ mW in weak inversion, and 13 Mbps/ mW in moderate inversion. Again the estimated speed applies to the decoder only, and not the interfaces. The interfaces of the Product decoder are the same as those of the component decoder. They therefore have the same limitations and should operate at the same speed as the interface for the component decoder, discussed in Section 9.2.

10.2.2 Performance.

Performance measurements for the Product decoder are shown in Figure 10.2.2. The decoder was measured with $I_U = 50\text{nA}$, at a speed of 788 kbps. To obtain the results of Figure 10.2.2, only a single bit-position is observed. Fifty errors are counted for each data point.

As explained in Section 9.3.4, the comparators used for the output interface have very low yield. For this reason, only the bit position with the best performance is shown. This measurement provides a reasonable verification of the decoder itself, based on the arguments outlined in Section 9.4.

To identify the best bit position, the transmitted codeword is varied randomly between all-zeros and all-ones. This makes it easy to distinguish between bits which are stuck at zero or one and those which are switching appropriately.

In addition to measured performance, Figure 10.2.2 shows the $d_{\min}$ asymptote, adjusted to account for the error probability of a fixed bit-position instead of the entire code. The multiplicity of minimum-weight error patterns with a fixed error position for the (16,11) component code is 35. In the Product code, this refers to the number of row error patterns for a fixed position. There are then 35 possible column error patterns, which may occur on one of four possible columns.

The total multiplicity is $35 \cdot 35 \cdot 4 = 4900$. Substituting this multiplicity into the $d_{\min}$ bound (Equation 2.5.1), we arrive at

$$P_e > \frac{16}{2 \cdot 256} \cdot 4900 \cdot \operatorname{erfc}\left(\sqrt{\frac{d_{\min} \cdot R \cdot E_b}{N_0}}\right). \tag{10.2.1}$$

The measured performance in Figure 10.2.2 is very close to this bound.

Also shown is the performance of a software simulation of an iterative product decoder. The software decoder was measured in precisely the same way as the decoder chip. Interestingly, for SNR less than 3dB the analog decoder outperforms the software decoder by more than 1dB. This is the first time that an apparent *implementation gain* has been observed in an analog decoder.

It is of course possible that the results shown in Figure 10.2.2 are biased or otherwise erroneous. The measurement interface has been thoroughly examined, and the author is quite confident of its correctness. For additional verification, the software decoder was substituted for the hardware interface within the same program used to measure the chip's performance. The results of this simulation were identical to the software decoder's performance, as reported in Figure 10.2.2.

The remaining possible cause of erroneous results is bias within the decoder itself. This would require a block-wise bias toward *both* the all-zero and all-one codewords. This is highly unlikely because there is nothing special about these two codewords in the circuit. The two codewords could easily be exchanged with another pair of codewords by inverting the sign of a subset of bits at the interfaces.

This label-flipping procedure rotates the entire code space without modifying the decoder circuit at all. There are 2^{10} such isomorphisms (one for each codeword/inverse pair). The probability that two codewords have a simultaneous bias in the decoder circuit, and that those two codewords happened to receive the labels all-zero and all-one in this implementation, seems vanishingly small.

Lastly, we note that there is an error floor in the measured results at a BER of approximately 10^{-5}. This error floor is likely caused by a constellation of systematic effects, including thermal noise and clock feed-through in the DAC interface, and leakage currents in the S/H array. Some of these effects were noted in Section 9.3.4, where they were found to cause artificial error floors in an uncoded interface test.

The observed error floor in this case is strongly influenced by S/H leakage. This is evident in a series of measurements shown in Figure 10.2.3. A set of four measured curves is shown, in which the bit rate was increased, beginning at 394 kbps, then 591 kbps, 675 kbps and finally 788 kbps. The corresponding sample rates are 833 kSps, 1.25 MSps, 1.43 MSps, and 1.67 MSps.

Each time the bit rate was increased, the error floor was seen to occur at a lower BER. With the current test interface, it is not possible to increase the speed beyond 1.67 MSps, because clock feed-through and ringing become dominant components in the DAC's output, so that the performance is actually made worse.

There is reason to expect the error floor to disappear as the speed is increased. In theory, the decoder can operate at several hundred megabits per second. It is probable that testing speeds less than 1 Mbps are simply too slow, and allow for additional data corruption.

Figure 10.2.3 also hints at the possibility that the "floors" are really "flares." This is evident in the first measured curve, which appears to floor at SNR=1.5dB, but then resumes its downward slope at 3dB. The error floor therefore takes on a "knee" shape. Similar phenomena may happen at the other tested speeds, but the tests were too time-consuming to verify this behavior at error rates below 10^{-5}.

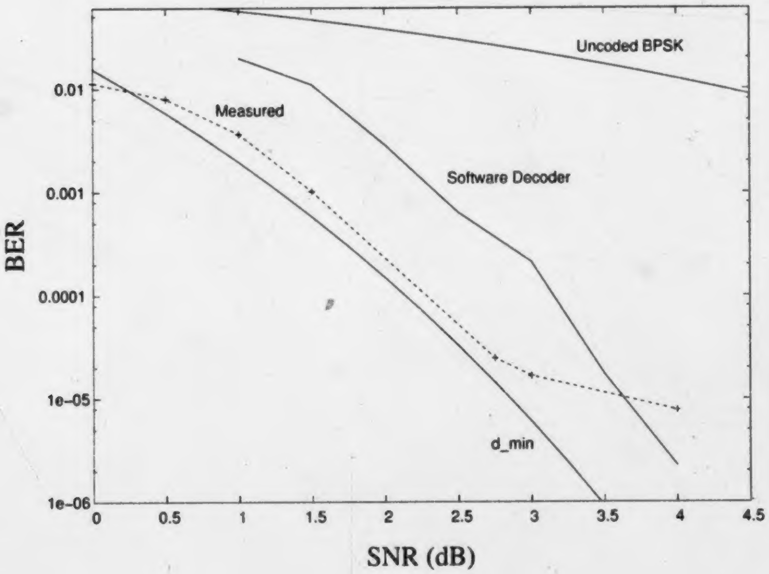


Figure 10.2.2: Test results for the Product decoder.

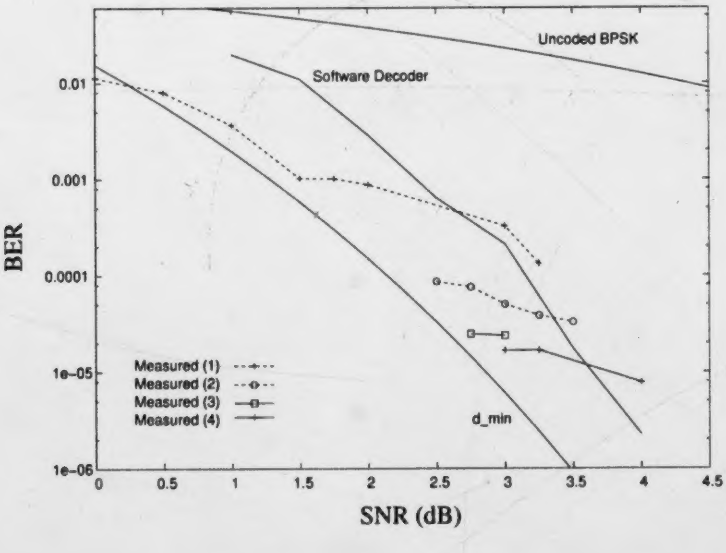


Figure 10.2.3: Error floors observed in measurements of the Product decoder.

Table 10.1: Summary of $(16,11)^2$ Product decoder characteristics. Speed and power measurements are estimated based on simulations, but were outside the accurately measurable range of available test equipment. "Tested speed" refers to the maximum reliable test speed of our interface.

Die Size	2.3 mm × 2.4 mm
Technology	0.18μm 6M 1.8V CMOS
Circuit Area (with interfaces)	4.0 mm^2
Decoder Area	2.85 mm^2
Transistor Size	1.15μm(W)×0.35μm(L)
Tested Speed	up to 2 Mbps
Core Decoder Power (est.)	861 μW @ $I_U = 100$nA 86.1 mW @ $I_U = 1\,\mu$A
Decoder Speed (est.)	5 Mbps @ $I_U = 100$nA 1 Gbps @ $I_U = 1\,\mu$A
Digital Power (est.)	6.84μW
Comparator Power (est.)	1.54μW/bit, avg.
S/H Power (est.)	1.8μW/bit, avg.
Total Power (est.)	1.72 mW, avg. @ $I_U = 100$nA

Chapter 11

Conclusions and Outlook.

The theme of this thesis, and its central conclusion, is that it is feasible to implement large-scale iterative decoders using analog computation in digital CMOS processes. These decoders can be directly interfaced with other receiver components for full or partial system-on-chip solutions.

It is in principle possible to replace any digital iterative decoder with a pin-for-pin analog replacement. There are several advantages to this transition:

- Analog decoders require dramatically less silicon area, and therefore dramatically lower cost.
- Analog decoders save substantial power. A conventional (digital) iterative decoder can consume power comparable to that of other major receiver components, often approaching 1 W. CMOS analog decoders typically operate at less than 100 mW.

These benefits notwithstanding, the semiconductor industry must respond to additional concerns, which affect the acceptability of a new technology:

- Designability – how much expertise and cost is required to design the components of the novel architecture?
- Manufacturability:
  - Yield – what are the design factors affecting device yield? How can they be improved? How do they scale?

- Performance margins – how do we define and verify the quality of a particular device? Is built-in-self-test (BIST) possible?

Some of these issues are isomorphic to existing mixed-signal design challenges. As shown in this thesis, the performance of an analog decoder depends most critically upon the interfaces. Issues of designability and manufacturability, as they relate to the input interface, are reducible to common design problems faced by DAC and ADC designers. The challenge posed by the output interface is to create a dense array of comparators with low offset variance. The same challenge is faced in the design of DRAM circuits.

We first address the issue of designability. One emphasis of the designs presented in this thesis is to demonstrate that complex analog decoders can be synthesized from a small number of simple standard cells. This is made especially clear by the design of the (16,11) Hamming decoder, which required only a few standard cells.

Software is available which translates an error control code's graph directly into a standard-cell schematic for an analog decoder [24]. Schematic-level design for the decoder itself can therefore be completely automated. With a suitable standard cell library, the decoder itself simply poses a challenge of place-and-route.

For most of the actual decoding circuits (which are referred to as the "lateral" processing stages in Section 7.2), the design needs are, therefore, not very different from those of a large digital circuit. With a few days' worth of instruction, any digital designer who is familiar with iterative decoders should be able to produce a successful, complete analog decoder design.

For the front-end circuits, the situation is slightly more complicated. As discussed in Section 7.2, "feed-forward" analog processing circuits are highly sensitive to mismatch. The sensitivity of these circuits is dictated by a simple formula (Equation 7.2.8, studied in more detail in [24]).

Conventional techniques for reducing mismatch, which are well-known to any mixed-signal designer, can easily reduce mismatch to within 1%. With such a small mismatch variance, the performance loss is always very small (as shown in Section

7.2.2).

Mixed-signal circuits, such as analog-to-digital converters (ADCs), often require mismatch variance in some transistors to be smaller than 1%. The front-end circuits to analog decoders consist primarily of differential pairs and current-mirror circuits, which are the most fundamental and well-understood components of analog and mixed-signal design.

Other issues affecting the front-end circuits include thermal noise, bandwidth, charge-injection, and drive strength. None of these issues are foreign to an ADC designer. Thus any experienced ADC designer should find a familiar set of problems in the design of front-end circuits for analog decoders. A design team consisting of experienced digital VLSI designers and mixed-signal designers should therefore be able to produce a complete analog decoder, with very little special training.

The next set of issues involve manufacturability, particularly yield. We know from the discussion in Section 7.1.2 that the yield of analog decoders is particularly sensitive to the variance of offsets in comparators used at the decoder's output. A large array of comparators is used, and each one must have an acceptably low input offset, or the entire chip must be rejected.

A similar problem arises in the design of DRAM circuits, where a large, dense array of latches (which are basically analog comparators) is needed to read stored digital values. While this thesis does not investigate the details of comparator designs, it appears that the problem of large, high-yield comparator arrays has already been addressed and mostly solved by DRAM designers.

If the comparators are known to provide sufficiently high yield, then the question of yield and performance margins is shifted to the decoder itself. As shown in Section 7.2, the performance of a large decoder is expected to be impacted very little, as long as mismatch is within easily achieved limits. The only assumption made in that analysis is that every transistor has a functioning gate. In effect, this reduces the problem of verification to the discovery of stuck-at faults. This is the same verification problem that arises in all digital circuits.

We therefore draw our final conclusions as follows. Analog iterative decoding circuits are not only feasible. They have been demonstrated on small and moderate scales. They are known, analytically, to be robust against the analog imperfections, including mismatch, which usually plague analog computational circuits.

The deployment of analog decoding circuits in the communications and semiconductor industries is also quite feasible. The problem of designing analog decoders is reducible to widely-used approaches for digital design. The interfaces needed for analog decoders are simple extensions of the circuits already widely used in ADC and DAC circuits, and in DRAM designs. There is thus no significant barrier to the designability of analog decoders.

Having addressed the issues of performance, scaling, designability, and manufacturability of practical analog decoders, the only remaining question is that of *verification* for analog decoders. Large-scale manufacturing relies on the possibility of built-in self test (BIST).

Based the density evolution analysis of Section 7.2.3, most of the BIST problem for analog decoders is reducible to one of digital verification. But the decoding circuits are analog in nature. Some method is needed to "digitally verify" these analog circuits by verifying the existence of every gate.

It has been shown by Haley et. al. that analog sum-product circuits can be easily "mode-switched" to make them temporarily behave as digital logic gates [40]. This approach can feasibly be used to create a digital-mode BIST for analog decoding circuits.

The benefits of analog decoders over their digital counterparts are now well understood. With a successful demonstration of a BIST approach for analog decoders, the major barriers to their deployment should be solved. While such a demonstration is a matter for future research, once it is achieved, it is reasonable to declare that analog decoding is a sufficiently mature technology to warrant industry attention.

Bibliography

[1] GNU scientific library. http://www.gnu.org/software/gsl/.

[2] Third Generation Partnership Project (3GPP). 3G TS 25.212, v3.5.0, Multiplexing and Channel Coding (FDD), Dec 2000.

[3] IEEE Std. 802.16a 2003. Amendment 2: Medium access control modifications and additional physical layer specifications for 2-11 Ghz, January 2003.

[4] A. Acampora and R. Gilmore. Analog Viterbi decoding for high speed digital satellite channels. *IEEE Transactions on Communications*, 26(10):1463–1470, October 1978.

[5] Comtech AHA. http://www.aha.com, 2004.

[6] Ibrahim Al-Mohondes and Mohamed Elmasry. A low-power 5 Mb/s Turbo decoder for third-generation wireless terminals. In *Proc. 2004 Canadian Conference on Electrical and Computer Engineering (CCECE'04)*, volume 4, pages 2387–2390, 2004.

[7] J.B. Anderson and S.M. Hladik. Tailbiting MAP decoders. *IEEE Journal on Selected Areas in Communications*, 16(2):297–302, February 1998.

[8] J. H. Atherton and H. T. Simmonds. An offset reduction technique for use with CMOS integrated comparators and amplifiers. *IEEE Journal of Solid-State Circuits*, 27(8):1168–1175, August 1992.

[9] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Transactions on Information Theory*, pages 284–287, March 1974.

[10] A.H. Banihashemi and S. Hemati. Decoding in optics. In *Proc. International Symposium on Information Theory*, page 231, June 2002.

[11] A.H. Banihashemi and S. Hemati. Analog min-log APP decoder. In *Proc. 2003 International Symposium on Information Theory (ISIT'03)*, June 2003.

[12] S. Benedetto and G. Montorsi. Unveiling Turbo codes: some results on parallel concatenated coding schemes. *IEEE Transactions on Information Theory*, 42(2):409–428, March 1996.

[13] Sergio Benedetto and Ezio Biglieri. *Principles of Digital Transmission with Wireless Applications*. Kluwer Academic Press, 1999.

[14] C. Berrou, P. Combelles, P. Penard, and B. Talibart. An IC for Turbo-codes encoding and decoding. In *Proc. 1995 IEEE International Solid-State Circuits Conference (ISSCC'95)*, pages 90–91, 1995.

[15] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon limit error-correcting coding and decoding: Turbo codes. In *Proc. 1993 International Communications Conference (ICC'93)*, pages 1064–1070, Geneva, Switzerland, May 1993.

[16] C. Berrou, A. Glavieux, and P. Thitimajshima. Near Shannon-limit error-correcting coding and decoding: Turbo codes. *IEEE Transactions on Communications*, 44(10):1261–1271, October 1996.

[17] M. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, C. Nicol, and R.-H. Yan. A unified Turbo / Viterbi channel decoder for 3GPP mobile wireless in 0.18μm CMOS. *Proc. 2002 IEEE International Solid State Circuits Conference (ISSCC'02)*, pages 90–91, February 2002.

[18] Andrew J. Blanksby and Chris J. Howland. A 690-mW 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder. *IEEE Journal of Solid-State Circuits*, 37(3):404–412, March 2002.

[19] A. R. Calderbank, G. D. Forney, and A. Vardy. Minimal tail-biting trellises: The Golay code and more. *IEEE Transactions on Information Theory*, pages 1435–1455, July 1999.

[20] Sae-Young Chung, G. David Forney Jr., Thomas J. Richardson, and Rüdiger Urbanke. On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit. *IEEE Communications Letters*, 5(2):58–60, February 2001.

[21] Sae-Young Chung, Thomas J. Richardson, and Rudiger Urbanke. Analysis of sum-product decoding of low-density parity-check codes using a Gaussian approximation. *IEEE Transactions on Information Theory*, pages 657–670, February 2001.

[22] J. H. Conway and N. J. A. Sloane. *Sphere-packings, lattices and groups*. Springer-Verglag, New York, 1988.

[23] T. M. Cover and J. Thomas. *Elements of Information Theory*. Wiley, 1991.

[24] Jie Dai. *Design Methodology for Analog VLSI Implementations of Error Control Decoders*. PhD thesis, University of Utah, 2001.

[25] Jie Dai, C.J. Winstead, C.J. Myers, R.R. Harrison, and C. Schlegel. Cell library for automatic synthesis of analog error control decoders. In *Proc. International Symposium on Circuits and Systems*, volume 4, pages IV–481 – IV–484, May 2002.

[26] A. Demosthenous and J. Taylor. A 100-Mb/s 2.8-V CMOS current-mode analog Viterbi decoder. *IEEE Journal of Solid-State Circuits*, 37(7):904–910, July 2002.

[27] P. Drennan and C. McAndrew. Understanding MOSFET mismatch for analog design. *IEEE Journal of Solid State Circuits*, 38(3):450–456, March 2003.

[28] P. Elias. Error-free coding. *IRE Trans. on Information Theory*, IT-4:29–37, September 1954.

[29] G. D. Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973.

[30] G. D. Forney. Coset codes – part II: binary lattices and related codes. *IEEE Transactions on Information Theory*, IT-34(5):1152–1187, September 1988.

[31] G.D. Forney. Codes on graphs: normal realizations. *IEEE Transactions on Information Theory*, pages 520–548, February 2001.

[32] Matthias Frey, Hans-Andrea Loeliger, Felix Lustenberger, Patrick Merkli, and Patrik Strebel. Analog-decoder experiments with subthreshold CMOS soft-gates. In *Proc. 2003 International Symposium on Circuits and Systems (ISCAS'03)*, pages 85–88, Bangkok, Thailand, May 2003.

[33] R. G. Gallager. *Low-Density Parity Check Codes*. MIT Press, Cambridge, MA, 1963.

[34] V. Gaudet. Toward Gigabit-per-second decoding. In *Proc. Analog Decoding Workshop*. Munich, June 2002.

[35] V. C. Gaudet and P. G. Gulak. A 13.3-Mb/s 0.35-μm CMOS analog turbo decoder IC with a configurable interleaver. *IEEE Journal of Solid-State Circuits*, 38(11):2010–2015, November 2003.

[36] J. Hagenauer, M. Moerz, and A. Schaefer. Analog decoders and receivers for high speed applications. In *Proc. Int. Zurich Seminar on Broadband Comm.*, pages 3-1–3-8, 2002.

[37] J. Hagenauer, E. Offer, and L. Papke. Iterative decoding of binary block and convolutional codes. *IEEE Transactions on Information Theory*, 42(2):429–445, March 1996.

[38] J. Hagenauer and M. Winklhofer. The analog decoder. *Proc. International Symposium on Information Theory*, August 1998.

[39] D. Haley, A. Grant, and J. Buetefuer. Iterative encoding of low-density parity-check codes. In *Proc. IEEE Globecom*, October 2002.

[40] Dave Haley, Chris Winstead, Christian Schlegel, and Alex Grant. An analog LDPC codec core. In *International Symposium on Turbo Codes*, pages 391–394. Brest, France, 2003.

[41] Dave Haley, Chris Winstead, Christian Schlegel, and Alex Grant. Architectures for error control in analog subthreshold CMOS. In *Australian Communication Theory Workshop*, 2003.

[42] S. Hong, J. Yi, and W. E. Stark. VLSI design and implementation of low-complexity adaptive Turbo-code encoder and decoder for wireless mobile communication applications. In *Proc. 1998 IEEE Workshop on Signal Processing Systems (SIPS'98)*, pages 233–242, October 1998.

[43] W. Huang, V. Igure, G. Rose, Y. Zhang, and M. Stan. Analog Turbo decoder implemented in SiGe BiCMOS technology. *40th DAC Student design contest*, 2003.

[44] W. Huang, V. Igure, G. Rose, Y. Zhang, and M. Stan. Analog Turbo decoder implemented in SiGe BiCMOS technology, July 2003. Available at http://www.ece.virginia.edu/hplp/turbo.html.

[45] Alexandre Graell i Amat, Sergio Benedetto, Guido Montorsi, Daniele Vogrig, Andrea Neviani, and Andrea Gerosa. An analog Turbo decoder for the UMTS standard. In *Proc. International Symposium on Information Theory*, 2004.

[46] Jagadeesh Kaza and Chaitali Chakrabarti. Design and implementation of low-energy Turbo decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(9):968–977, September 2004.

[47] Hyongsuk Kim, Hongrak Son, T. Roska, and L. O. Chua. Very high speed Viterbi decoder with circularly connected analog CNN cell array. *Proc. 2004 Int'l Symposium on Circuits and Systems (ISCAS '04)*, 3:III – 97–100, May 2004.

[48] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, February 2001.

[49] Frank Kschischang and Vladislav Sorokine. On the trellis structure of block codes. *IEEE Transactions on Information Theory*, 41(6):1924–1937, November 1995.

[50] G.P. Lepage. A new algorithm for adaptive multidimensional integration. *Journal of Computational Physics*, pages 192–203, 1978.

[51] Shih-Chii Liu, Jörg Kramer, Giacomo Indiveri, Tobias Delbrück, and Rodney Douglas. *Analog VLSI: Circuits and Principles*. MIT Press, 2002.

[52] H. A. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarkoy. Probability propagation and decoding in analog VLSI. *IEEE Transactions on Information Theory*, 47(2):837–843, February 2001.

[53] F. Lustenberger. *On the Design of Analog VLSI Iterative Decoders*. PhD thesis, Swiss Federal Institute of Technology, 2000.

[54] F. Lustenberger, M. Helfenstein, G. S. Moschytz, H. A. Loeliger, and F. Tarkoy. All analog decoder for (18,9,5) tail-biting trellis code. In *Proc. European Solid-State Circuits Conference (ESSCIRC)*, pages 362–365, Sept. 1999.

[55] F. Lustenberger and H. A. Loeliger. On mismatch errors in analog-VLSI error correcting decoders. In *Proc. International Symposium on Circuits and Systems*, May 2001.

[56] D. J. C. MacKay. Good error-correcting codes based on very sparse matrices. *IEEE Transactions on Information Theory*, 45:399–431, Mar 1999.

[57] T. W. Matthews and R. R. Spencer. An integrated analog CMOS Viterbi detector for digital magnetic recording. *IEEE Journal of Solid-State Circuits*, 28(12):1294–1302, December 1993.

[58] Carver Mead. *Analog VLSI and Neural Systems*. Addison-Wesley, 1989.

[59] M. Moerz, T. Gabara, R. Yan, and J. Hagenauer. An analog 0.25μm BiCMOS tailbiting MAP decoder. In *International Solid State Circuits Conference*, pages 356–357, February 2000.

[60] M. Moerz, A. Schaefer, and E. Offer. Analog decoding of high rate tailbiting codes using the dual trellis. In *Proc. International Symposium on Information Theory*, page 331, 2001.

[61] A. Mondragon-Torres and E. Sanchez-Sinencio. Floating gate analog implementation of the additive soft-input soft-output decoding algorithm. In *Proc. International Symposium on Circuits and Systems*, pages 89–92, May 2002.

[62] A. F. Mondragon-Torres, E. Sanchez-Sinencio, and K. R. Narayanan. Floating-gate analog implementation of the additive soft-input soft-output decoding algorithm. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 50(10):1256–1269, oct 2003.

[63] R. S. Muller and T. I. Kamins. *Device electronics for integrated circuits*. John Wiley and Sons, 1977.

[64] V. Gaudet N. Nguyen, C. Winstead and C. Schlegel. A 0.8V CMOS analog decoder for an (8,4,4) extended Hamming code. In *Proc. International Symposium on Circuits and Systems*, volume 1, pages I – 1116–1119. Vancouver, Canada, May 2004.

[65] M. Pelgrom, A. Duinmaijer, and A. Welbers. Matching properties of MOS transistors. *IEEE Journal of Solid State Circuits*, 24(5):1433–1440, October 1989.

[66] M. Perenzoni, A. Gerosa, and A. Neviani. Analog CMOS implementation of Gallager's iterative decoding algorithm applied to a block Turbo code. In *Proc. 2003 International Symposium on Circuits and Systems (ISCAS '03)*, volume V, pages V – 813–816, May 2003.

[67] R. Pyndiah. Near optiumum decoding of product codes: Block Turbo codes. *IEEE Transactions on Information Theory*, 42(8), August 1998.

[68] Jan M. Rabaey, Anantha Chandrakasan, and Borivoje Nikolic. *Digital Integrated Circuits*. Prentice Hall, 2nd edition, 2002.

[69] Thomas J. Richardson, M. Shokrollahi, and Rudiger Urbanke. Design of capacity-approaching irregular low-density parity-check codes. *IEEE Trans. Inform. Theory*, pages 619–637, February 2001.

[70] Thomas J. Richardson and Rudiger Urbanke. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Trans. Inform. Theory*, pages 599–618, February 2001.

[71] L. D. Rudolph, C. R. P. Hartmann, T.-Y. Hwang, and N. Q. Duc. Algebraic analog decoding of linear binary codes. *IEEE Transactions on Information Theory*, IT-25(4):430–440, July 1979.

[72] A. Schaefer, M. Moerz, J. Hagenauer, A. Sridharan, and D. J. Costello Jr. Analog rotating ring decoder for an LDPC convolutional code. *Int'l Information Theory Workshop*, 31:226–229, April 2003.

[73] C. Schlegel and L. C. Pérez. *Trellis and Turbo Coding*. IEEE Press, 2004.

[74] E. Seevinck, E. Vittoz, M. du Plessis, T.-H. Joubert, and W. Beetge. CMOS translinear circuits for minimum supply voltage. *IEEE transactions on circuits and systems II*, 47(12):1560–1564, December 2000.

[75] T. Serrano-Gotarredona, B. Linares-Barranco, and A. G. Andreou. A general translinear principle for subthreshold MOS transistors. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 46(5):607–616, May 1999.

[76] M. H. Shakiba, D. A. Johns, and K. W. Martin. BiCMOS circuits for analog Viterbi decoders. *IEEE Transactions on Circuits and Systems II*, 45:1527–1537, December 1998.

[77] M.H. Shakiba, D.A. Johns, and K.W. Martin. An integrated 200MHz 3.3V BiCMOS class-IV partial-response analog Viterbi decoder. (1):61–75, January 1998.

[78] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, July 1948.

[79] J.-H. Shieh, M. Patel, and B. J. Sheu. Measurement and analysis of charge injection in MOS analog switches. *IEEE Journal of Solid-State Circuits*, 22:277–281, April 1987.

[80] R. R. Spencer and P. J. Hurst. Analog implementation of sampling detectors. *IEEE Transactions on Magnetics*, 27(6):4516–4521, November 1991.

[81] Stephan ten Brink. A rate one-half code for approaching the Shannon limit by 0.1dB. *IEE Electronics Letters*, 36(15):1293–1294, July 2000.

[82] E. Vittoz and J. Fellrath. CMOS analog integrated circuits based on weak-inversion operation. *IEEE Journal of Solid-State Circuits*, SC-12:224–231, 1977.

[83] Xiao-An Wang and S. B. Wicker. An artificial neural net Viterbi decoder. *IEEE Transactions on Communications*, 44(2):165–171, February 1996.

[84] N. Wiberg, H. A. Loeliger, and R. Kotter. Codes and iterative decoding on general graphs. *European Transactions on Telecommunications*, pages 513–525, Sept./Oct. 1995.

[85] C. Winstead, J. Die, R. Harrison, C. J. Myers, and C. Schlegel. Analog decoding of product codes. In *Information Theory Workshop*, pages 131–133, August 2001.

[86] C. Winstead, J. Die, W.J. Kim, S. Little, Y.-B. Kim, C. J. Myers, and C. Schlegel. Analog MAP decoder for (8,4) Hamming code in subthreshold CMOS. In *Advanced Research in VLSI*, pages 132–147, March 2001.

[87] C. Winstead, J. Die, S. Yu, R. Harrison, C. J. Myers, and C. Schlegel. Analog MAP decoder for (8,4) Hamming code in subthreshold CMOS. In *Proc. International Symposium on Information Theory*, page 330, June 2001.

[88] C. Winstead and C. Schlegel. Importance sampling for SPICE-level verification of analog decoders. In *Proc. International Symposium on Information Theory*, page 103. Yokohama, June 2003.

[89] C. Winstead and C. Schlegel. Density evolution analysis of device mismatch in analog decoders. In *Proc. International Symposium on Information Theory*. Chicago, June 2004.

[90] Chris Winstead, Jie Dai, Shuhuan Yu, Reid Harrison, Chris J. Myers, and Christian Schlegel. CMOS analog decoder for (8,4) Hamming code. *IEEE Journal of Solid-State Circuits*, pages 122–131, January 2004.

[91] Chris Winstead, Vincent C. Gaudet, and Christian Schlegel. Analog iterative decoding of error control codes. In *Proc. 2003 IEEE Canadian Conference on Electrical Engineering (CCECE '03)*, volume 2, pages 1539–1542, 2003.

[92] Chris Winstead, Nhan Nguyen, Vincent C. Gaudet, and Christian Schlegel. Low-voltage CMOS circuits for analog decoders. In *International Symposium on Turbo Codes*, pages 271–274. Brest, France, September 2003.

[93] Koon-Lun Jackie Wong and Chih-Kong Ken Yang. Offset compensation in comparators with minimum input-referred supply noise. *IEEE Journal of Solid-State Circuits*, 39(5):837–840, May 2004.

[94] A. Worthen, S. Hong, R. Gupta, and W. Stark. Performance optimization of VLSI transceivers for low-energy communications systems. *Military Communications Conference*, November 1999.

[95] A. Xotta, D. Vogrig, A. Gerosa, A. Neviani, A. Graell-Amat, G. Montorsi, M. Bruccoleri, and G. Betti. An all-analog CMOS implementation of a Turbo decoder for hard-disk drive read channels. *Proc. International Symposium on Circuits and Systems*, pages 69–72, 2002.

[96] Shuhuan Yu. *Design and test of error control decoders in analog CMOS*. PhD thesis, University of Utah, 2004.